1 Préambule

Objectif :

L'objectif de ce projet est de résoudre en trois étapes le défi :

https://pydefis.callicode.fr/defis/C22_BwaCode02/txt

On vous fournit une archive `materiel.zip` *avec :*

1. *Un squelette de code Python* `projet_lapins_cretins_final.py` *à compléter ; contenant les mots proposables, extrait directement du dépôt* https://framagit.org/JonathanMM/sutom *du projet officiel ;*
2. *Un fichier texte* `message_lapin.txt` *contenant le message en codage lapins crétins qui est la donnée d'entrée du défi.*

La remise du projet s'effectuera en trois étapes dans les espaces de dépôts correspondants du cours Moodle :

- ***Étape 1 :*** *le jeudi 20 octobre*
- ***Étape 2 :*** *le jeudi 17 novembre*
- ***Étape 3 :*** *le jeudi 8 décembre*

Certains exercices faits et corrigés en classe ne seront pas pris en compte dans l'évaluation mais sont utiles à l'avancement du projet. Chaque étape sera évaluée séparément. Lorsque vous commencerez une nouvelle étape, vous disposerez du corrigé de l'étape précédente.
Chaque étape de remise doit s'effectuer conformément au cahier des charges suivant :

- ☞ *Les scripts fournis dans l'archive* `projet_lapins_cretins_final.py` *doivent être complétés à partir des squelettes de code fournis et rassemblés dans une archive* `zip` *nommée* `eleve_etape_num.zip` *déposée dans l'espace prévu sur le Moodle du cours (accessible via l'ENT).*
- ☞ *Chaque fonction doit être documentée avec une* `docstring`*.*
- ☞ *Les parties les moins évidentes du code doivent être commentées de façon pertinente.*
- ☞ *Le code produit doit être confronté aux tests unitaires fournis dans les deux scripts de test. Les tests échoués doivent être mentionnés en commentaire dans les scripts de tests.*

Thèmes du programme abordés :

- *Types simples : codage des entiers et des caractères ;*
- *Types de données construits : tableaux, chaînes de caractères ;*
- *Noyau d'un langage de programmation : boucles, tests, fonctions ;*
- *Tester : tests unitaires de fonctions.*

2 Étape 1 : codage de caractères par un entier écrit en base trois

Méthode *Extraire les chiffres d'un entier naturel*

Pour écrire les nombres entiers naturels, nous utilisons un système positionnel en base dix :

$$248 = 2 \times 10^2 + 4 \times 10^1 + 8 \times 10^0$$

On distingue les chiffres par l'exposant de la puissance de 10 qui s'appelle leur poids : 2 est le chiffre de poids le plus fort et 8 celui de poids le plus faible.
Pour extraire les chiffres d'un entier naturel, il suffit d'effectuer des *divisions euclidiennes en cascade* : le reste nous donne un chiffre et on poursuit avec le quotient. On récupère les chiffres par poids croissant et on s'arrête lorsque le quotient est nul

$$\begin{array}{r|l|l|l}
248 & 10 & & \\
\cline{2-2}
\mathbf{8} & 24 & 10 & \\
\cline{3-3}
 & \mathbf{4} & 2 & 10 \\
\cline{4-4}
 & & \mathbf{2} & 0
\end{array}$$

Exercice 1 *Fait en classe, non évalué*

On fournit dans `projet_lapins_cretins_final.py` une implémentation de l'algorithme donnant le tableau des chiffres en base dix d'un entier naturel :

```
def chiffres_base_dix(n):
    """
    Description et postcondition :
            Renvoie les chiffres de la représentation de l'entier n >= 0
            en base dix sous la forme d'un tableau avec les chiffres
            par poids décroissant de gauche à droite
    Paramètres:
             n est un entier
    Préconditions:
            n >= 0
    Retour: un tableau de type list
    """
    t = []
```

```
    while n >= 10:
        c  = n % 10
        t = [c] + t
        n = n // 10
    t = [n] + t
    return t

def test_chiffres_base10():
    assert chiffres_base_dix(0) == [0], "echec sur le cas
         chiffres_base_dix(0)"
    assert chiffres_base_dix(8) == [8], "echec sur le cas
         chiffres_base_dix(8)"
    assert chiffres_base_dix(10) == [1, 0], "echec sur le cas
         chiffres_base_dix(10)"
    assert chiffres_base_dix(87) == [8, 7], "echec sur le cas
         chiffres_base_dix(87)"
    assert chiffres_base_dix(807) == [8, 0, 7], "echec sur le cas
         chiffres_base_dix(807)"
    print("tests réussis")
```

1. Quel est le rôle des différents éléments présents dans chaîne de documentation ou `docstring` de la fonction ?

.....

.....

.....

.....

.....

.....

.....

2. Quel est le type de la variable `t` ? Comment ajouter un élément à `t` ?

.....

.....

.....

3. Faire un tableau d'évolution des variables `t`, `n` et `c` lors de l'exécution de `chiffres_base_dix(605)`.

.....

.....

.....

.....

.....

.....

.....

4. Qu'est-ce qu'un cas de test ? un jeu de tests ? un test unitaire ? Quelle est la syntaxe de l'instruction `assert` ? Compléter `test_chiffres_base10()` avec d'autres cas de tests ?

.....

.....

.....

.....

.....

.....

.....

Méthode *Changement de base*

On peut changer la base de notre système positionnel d'écriture des entiers naturels. En base dix, on a besoin de dix chiffres compris entre 0 et 9. En base trois, on va utiliser trois chiffres compris entre 0 et 2. Par exemple le nombre 21 s'écrit $(210)_3$ en base trois car :

$$21 = 2 \times 3^2 + 1 \times 3 + 0 \times 3^0$$

Pour obtenir les chiffres en base trois d'un entier naturel, on peut appliquer l'algorithme des *divisions euclidiennes en cascade.*

$$\begin{array}{r|l}
21 & 3 \\ \cline{2-2}
\mathbf{0} & 7 \;\; \begin{array}{|l} 3 \\ \hline \end{array} \\ \cline{2-2}
 & \mathbf{1} \;\; 2 \;\; \begin{array}{|l} 3 \\ \hline \end{array} \\ \cline{3-3}
 & \qquad \mathbf{2} \;\; 0
\end{array}$$

On obtient d'abord les chiffres de poids faibles.

Exercice 2 *Fait en classe non évalué*

Dans `projet_lapins_cretins_final.py`, compléter la fonction `decimal_vers_ternaire(n)` en respectant la spécification donnée dans sa `docstring`. Vérifier que le jeu de tests unitaires fourni est validé en exécutant `test_decimal_vers_ternaire()` dans la console interactive.

Exercice 3 *Travail personnel évalué*

Dans `projet_lapins_cretins_final.py`, compléter la fonction `ternaire_vers_decimal(tab)` en respectant la spécification donnée dans sa `docstring`. Vérifier que le jeu de tests unitaires fourni est validé en exécutant `test_ternaire_vers_decimal()` dans la console interactive.

Méthode *Codage des caractères*

La mémoire de l'ordinateur n'étant remplie que de 0 ou de 1, les caractères sont aussi codées numériquement par des entiers écrits en base deux dans la mémoire.
Une **table de caractères** associe à chaque caractère un **point de code** ou **ordinal**. La plupart des tables de caractères actuelles associent encore aux caractères alphabétiques non accentués et à l'espace leur ordinal de la table ASCII définie dans les années 1960.
La fonction Python `ord` permet d'obtenir l'ordinal d'un caractère et `chr` est sa fonction réciproque.

```
>>> ord('A')
65
>>> chr(65)
'A'
>>> ord(' ')
32
>>> chr(32)
' '
```

Exercice 4 *Fait en classe non évalué*

Dans `projet_lapins_cretins_final.py`, compléter la fonction `generer_symboles()` en respectant la spécification donnée dans sa `docstring`. Vérifier que le jeu de tests unitaires fourni est validé en exécutant `test_generer_symboles()` dans la console interactive.

Méthode *Séquences de type `list` ou `str`*

Les tableaux de type `list` et les chaînes de caractères de type `str` sont des conteneurs d'éléments indexés par leur position. Les index commencent à 0 et sont incrémentés avec un pas de 1.

- ☞ La longueur d'une séquence `seq` s'obtient avec `len(seq)`.
- ☞ On accède à l'élément d'index `k` (s'il est entre 0 et `len(seq)` d'une séquence `seq` par `seq[k]` :

```
>>> seq = [17, 2, -1]
>>> len(seq)
3
>>> seq[0]
17
>>> seq[2]
```

```
-1
>>> seq[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> seq = 'bac'
>>> seq[1]
'a'
```

☞ On peut parcourir une séquence :

- soit par index :

```
>>> seq = [17, 2, -1]
>>> for k in range(len(seq)):
...     print(k)
...
0
1
2
```

- soit par élément :

```
>>> seq = [17, 2, -1]
>>> for e in seq:
...     print(e)
...
0
1
2
>>> for c in 'bac':
       print(e)
...
b
a
c
```

Exercice 5 *Fait en classe non évalué*

Dans `projet_lapins_cretins_final.py`, compléter la fonction `index_dans_sequence(elt, seq)` en respectant la spécification donnée dans sa `docstring`. Vérifier que le jeu de tests unitaires fourni est validé en exécutant `test_index_dans_sequence()` dans la console interactive.

Exercice 6 ***Travail personnel évalué***

1. Dans `projet_lapins_cretins_final.py`, compléter la fonction

   `codage_texte_vers_ternaire(texte, symb)` en respectant la spécification donnée dans sa `docstring`

   Valider le jeu de tests unitaires fourni en exécutant `test_codage_texte_vers_ternaire()` dans la console interactive.

2. Dans `projet_lapins_cretins_final.py`, compléter la fonction

   `decodage_ternaire_vers_texte(texte, symb)` en respectant la spécification donnée dans sa `docstring`.

   Valider le jeu de tests unitaires fourni en exécutant `test_decodage_ternaire_vers_texte()` dans la console interactive.

3 Étape 2 : codage de caractères dans le code des lapins crétins

Exercice 7 ***Fait en classe non évalué***

1. Dans `projet_lapins_cretins_final.py`, compléter la fonction

   `ternaire_vers_lapin(tab, traducteur)` en respectant la spécification donnée dans sa `docstring`.

   Valider le jeu de tests unitaires fourni en exécutant `test_ternaire_vers_lapin()` dans la console interactive.

2. Dans `projet_lapins_cretins_final.py`, compléter la fonction

   `lapin_vers_ternaire(tab, traducteur)` en respectant la spécification donnée dans sa `docstring`.

   Valider le jeu de tests unitaires fourni en exécutant `test_lapin_vers_ternaire()` dans la console interactive.

Exercice 8 ***Travail personnel évalué***

1. Dans `projet_lapins_cretins_final.py`, compléter la fonction

   `codage_ternaire_vers_lapin(code_ternaire, traducteur)` en respectant la spécification donnée dans sa `docstring`.

   Valider le jeu de tests unitaires fourni en exécutant `test_codage_ternaire_vers_lapin()` dans la console interactive.

2. Dans `projet_lapins_cretins_final.py`, compléter la fonction

   `decodage_lapin_vers_ternaire(code_lapin, traducteur)` en respectant la spécification donnée dans sa `docstring`.

   Valider le jeu de tests unitaires fourni en exécutant `test_decodage_lapin_vers_ternaire()` dans la console interactive.

Exercice 9 *Travail personnel évalué*

1. Dans `projet_lapins_cretins_final.py`, compléter la fonction

   `codage_texte_vers_lapin(texte, symb, traducteur)` en respectant la spécification donnée dans sa `docstring`.

   Valider le jeu de tests unitaires fourni en exécutant `test_codage_texte_vers_lapin()` dans la console interactive.

2. Dans `projet_lapins_cretins_final.py`, compléter la fonction

   `decodage_lapin_vers_texte(code_lapin, symb, traducteur)` en respectant la spécification donnée dans sa `docstring`.

   Valider le jeu de tests unitaires fourni en exécutant `test_decodage_lapin_vers_ternaire()` dans la console interactive.

4 Étape 3 : résolution du défi

Exercice 10 *Fait en classe non évalué*

Dans `projet_lapins_cretins_final.py`, compléter la fonction
`permutations(tab)` en respectant la spécification donnée dans sa `docstring`.
Valider le jeu de tests unitaires fourni en exécutant `test_permutations()` dans la console interactive.

Exercice 11 *Travail personnel évalué*

1. Dans `projet_lapins_cretins_final.py`, compléter la fonction

   `decoupage_bloc3(texte)` en respectant la spécification donnée dans sa `docstring`.

   Valider le jeu de tests unitaires fourni en exécutant `test_decoupage_bloc3()` dans la console interactive.

2. Dans `projet_lapins_cretins_final.py`, compléter la fonction

   `liste_blocs_vers_texte(liste_blocs)` en respectant la spécification donnée dans sa `docstring`.

   Valider le jeu de tests unitaires fourni en exécutant `test_liste_blocs_vers_texte()` dans la console interactive.

3. Exécuter `solution_pydefis("message_lapin.txt", "decodages_lapin.txt")`.

   Quelle est la réponse au défi ?