Exercice 1. Fonctions mystères

On considère les fonctions `g` et `h` suivantes :

```
def g():
    a = 1
    while a < 100:
        a = 67 + (-1)**a * (a//2)
    return a
```

```
def h():
    a = 1
    while a < 100:
        a = 4 + (-1)**a * (a//2)
    return a
```

1. Sans utiliser Python, prévoir la valeur de `g()`.
2. Idem pour la fonction `h`.

Exercice 2. Préfixes, facteurs et sous-mots

Soient `s1` et `s2` deux chaînes de caractères. On dit que `s1` est un ***préfixe*** de `s2` s'il existe une chaîne de caractères `s3` telle que `s2 = s1 + s3`. Par exemple, `"abcd"` possède 5 préfixes :

`""`, `"a"`, `"ab"`, `"abc"`, `"abcd"`.

1. À l'aide d'une boucle `while`, écrire une fonction « `est_prefixe(s1: string, s2: string) -> bool` » qui renvoie `True` lorsque `s1` est un préfixe de `s2` et `False` sinon. Tester avec les exemples du tableau (voir la fin de l'exercice).

Soient `s1` et `s2` deux chaînes de caractères. On dit que `s1` est un ***facteur*** de `s2` s'il existe deux chaînes de caractères `s3` et `s4` telles que `s2 = s3 + s1 + s4`. Ainsi, si `s1` est un préfixe de `s2`, c'est aussi un facteur. Par exemple, `"abcd"` possède 11 facteurs :

`""`, `"a"`, `"b"`, `"c"`, `"d"`, `"ab"`, `"bc"`, `"cd"`, `"abc"`, `"bcd"`, `"abcd"`.

2. À l'aide d'une boucle `while` et de la fonction précédente, écrire une fonction « `est_facteur(s1: string, s2: string) -> bool` » qui renvoie `True` lorsque `s1` est un facteur de `s2` et `False` sinon. Tester avec les exemples du tableau (voir la fin de l'exercice).

Soient `(s1, s2)` deux chaînes de caractères et `(n1, n2)` leurs tailles respectives. On dit que `s1` est un ***sous-mot*** de `s2` s'il existe $0 \leqslant i_0 < i_1 < \ldots < i_{n1-1} \leqslant n2 - 1$ tels que :

$$s1[0] = s2[i_0] \qquad s1[1] = s2[i_1] \qquad \ldots \qquad s1[n1-1] = s2[i_{n1-1}]$$

Ainsi, si `s1` est un facteur de `s2`, c'est aussi un sous-mot. Par exemple, `"abcd"` possède 16 sous-mots : ses 11 facteurs et

`"ac"`, `"ad"`, `"bd"`, `"abd"`, `"acd"`.

3. À l'aide d'une boucle `while`, écrire une fonction de signature « `est_sous_mot(s1: string, s2: string) -> bool` » qui renvoie `True` lorsque `s1` est un sous-mot de `s2` et `False` sinon. Tester avec les exemples du tableau (voir la fin de l'exercice).

s1	"Po"	"inc"	"Pic"	"Poinca"	"Poincaré"	"Poinca"
s2	"Poinca"	"Poinca"	"Poinca"	"Poinca"	"Poinca"	""
est_prefixe(s1, s2)	True	False	False	True	False	False
est_facteur(s1, s2)	True	True	False	True	False	False
est_sous_mot(s1, s2)	True	True	True	True	False	False

Exercice 3. Persistance multiplicative

Dans cet exercice, attention de ne pas confondre les notions d'« entier » et de « chiffre ». Un entier est un élément de $\mathbb{N}$, un chiffre est un élément de `{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}`.

Soit `n` $\in \mathbb{N}$ un entier naturel. On s'intéresse à l'opération consistant à multiplier les chiffres de `n` entre eux. Par exemple, lorsqu'on multiplie les chiffres de `15 638`, on obtient `1` $\times$ `5` $\times$ `6` $\times$ `3` $\times$ `8 = 720` et lorsqu'on multiplie les chiffres de `720` on obtient `7` $\times$ `2` $\times$ `0 = 0`.

1. À l'aide d'une boucle `while`, écrire une fonction de signature « `prod_chiffres(n: int) -> int` » qui renvoie le produit des chiffres de `n`. Votre fonction déclenchera une erreur si `n < 0`. Par exemple :

`n`	0	1	6	473	678	720	15638
`prod_chiffres(n)`	0	1	6	84	336	0	720

La ***persistance multiplicative*** d'un entier est le nombre d'itérations nécessaires pour obtenir un résultat avec un seul chiffre. Par exemple, la persistance multiplicative de `15 638` est `2`.

2. (a) Quelle est la persistance multiplicative de 678 ?
   (b) À l'aide d'une boucle `while`, écrire une fonction de signature « `pers_mult(n: int) -> int` » qui renvoie la persistance multiplicative de `n`. Votre fonction déclenchera une erreur si `n < 0`. Par exemple :

`n`	0	1	6	473	678	720	15638
`pers_mult(n)`	0	0	0	3	4	1	2

Pour tout `k` $\in \mathbb{N}$, on note $\mathtt{N}_{\mathtt{k}}$ le plus petit entier naturel dont la persistance multiplicative vaut `k`. Par exemple, $\mathtt{N}_0 = 0$ et $\mathtt{N}_1 = 10$.

3. (a) Sur feuille, déterminer $\mathtt{N}_2$ et $\mathtt{N}_3$.
   (b) À l'aide d'une boucle `while`, écrire une fonction de signature « `get_Nk(k: int) -> int` » qui renvoie $\mathtt{N}_{\mathtt{k}}$. Tester votre fonction pour `k` $\in [\![0\,;9]\!]$
   (c) **(facultatif, très difficile)** Améliorer la fonction précédente pour calculer $\mathtt{N}_{10}$ et $\mathtt{N}_{11}$.
   Remarque : personne n'a jamais trouvé de nombre dont la persistance multiplicative est supérieure ou égale à `12` (on ne sait pas si un tel nombre existe).

Exercice 4. Conjecture de Syracuse

Soit $N \in \mathbb{N}^*$ un entier strictement positif. On définit la suite de Syracuse $(u_n)_{n\in\mathbb{N}}$ associée à N de la manière suivante :

$$\begin{cases} u_0 = N \\ u_{n+1} = \dfrac{u_n}{2} & \text{si } u_n \text{ est pair} \\ u_{n+1} = 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$$

Il est conjecturé (mais ça n'est pas prouvé) que pour tout entier initial N, il existe un naturel n tel que $u_n = 1$, $u_{n+1} = 4$, $u_{n+2} = 2$, $u_{n+3} = 1$, $u_{n+4} = 4$, ... (la suite est donc périodique à partir du rang n). On appelle :

- ***Temps de vol*** le plus petit entier n tel que $u_n = 1$.
- ***Temps de vol en altitude*** le plus petit entier n tel que $u_{n+1} < u_0$.
- ***Altitude maximale*** la valeur maximale de la suite $(u_n)_{n\in\mathbb{N}}$.

1. Soit $n \in \mathbb{N}$ un entier, on suppose qu'on a déjà calculé l'entier u_n. Écrire une fonction `suivant` qui prend en entrée u_n et renvoie u_{n+1}. Votre fonction déclenchera une erreur si l'entier donné en entrée est négatif ou nul. Par exemple :

`suivant(0)` déclenche une erreur,
`suivant(15)` vaut `46` (et pas `46.0`),
`suivant(82)` vaut `41` (et pas `41.0`).

Dans les questions qui suivent, on pensera à utiliser la fonction `suivant` pour éviter de réécrire plusieurs fois le même code.

2. Écrire une fonction `temps_vol` qui prend en entrée N et renvoie le temps de vol de la suite. Votre fonction déclenchera une erreur si $N \leqslant 0$. Par exemple :

   `temps_vol(-5)` déclenche une erreur, `temps_vol(1)` vaut `0`,
   `temps_vol(2)` vaut `1`, `temps_vol(15)` vaut `17`,
   `temps_vol(27)` vaut `111`.

3. Écrire une fonction `temps_vol_altitude` qui prend en entrée N et renvoie le temps de vol en altitude de la suite. Votre fonction déclenchera une erreur si $N \leqslant 1$. Par exemple :

   `temps_vol_altitude(1)` déclenche une erreur, `temps_vol_altitude(2)` vaut `0`,
   `temps_vol_altitude(19)` vaut `5`, `temps_vol_altitude(31)` vaut `90`.

4. Écrire une fonction `altitude_max` qui prend en entrée N et renvoie l'altitude maximale de la suite. Votre fonction déclenchera une erreur si $N \leqslant 0$. Par exemple :

   `altitude_max(0)` déclenche une erreur, `altitude_max(1)` vaut `4`,
   `altitude_max(21)` vaut `64`, `altitude_max(83)` vaut `9232`.

Exercices à rendre au plus tard le 10/11/2024 à 20h

Dans ces exercices, vous n'êtes pas obligés d'utiliser des boucles `while`.

Exercice 5. Méthode `join` pour les chaînes de caractères

Écrire une fonction de signature « `join(sep: str, L: list[str]) -> str` » qui renvoie la concaténation des éléments de `L` séparés par `sep`. Par exemple :

`join(" ", ["J'aime","l'informatique"])`	vaut	`"J'aime l'informatique"`
`join("--", ["toto","titi","tata"])`	vaut	`"toto--titi--tata"`
`join("", ["Mots", "Sans", "Espace"])`	vaut	`"MotsSansEspace"`
`join("...", ["a","b","c","d","e","f"])`	vaut	`"a...b...c...d...e...f"`
`join("--", [])`	vaut	`""`

Remarque. En réalité, `join` existe déjà en Python : il suffit d'écrire `sep.join(L)`. Bien sûr vous n'avez pas le droit d'utiliser cette méthode dans cet exercice.

Remarque. Si on définit « `s = join(" ", ["J'aime","l'informatique"])` », alors :

→ `s[0]` vaut `'J'` et pas `'"'`.

→ `s[-1]` vaut `'e'` et pas `'"'`.

→ `len(s)` vaut 21 et pas 23.

Exercice 6. Méthode `split` pour les chaînes de caractères

Écrire une fonction de signature « `split(s: str) -> list[str]` » qui renvoie la liste constituée des mots de `s`. Par exemple :

`split("Ceci est un test")`	vaut	`["Ceci", "est", "un", "test"]`
`split("Avec six espaces")`	vaut	`["Avec", "six", "espaces"]`
`split("UnSeulMot")`	vaut	`["UnSeulMot"]`
`split("  test  ")`	vaut	`["test"]`
`split("")`	vaut	`[]`

Si besoin, on pourra lire les indications ci-dessous.

Remarque. En réalité, `split` existe déjà en Python : il suffit d'écrire `s.split()`. Bien sûr vous n'avez pas le droit d'utiliser cette méthode dans cet exercice.

Indications (essayez de résoudre l'exercice sans lire ce qui suit).

1. À l'aide d'une boucle `while`, écrire une fonction de signature « `debut_mot(s: str, i: int) -> int` » qui renvoie le plus petit indice `j` $\geqslant$ `i` tel que `s[j]` n'est pas un espace. Si pour tout `j` $\geqslant$ `i`, le caractère `s[j]` est un espace, votre fonction renverra `len(s)`. Par exemple, avec `s = " a bc d "`, on obtient :

`i`	0	1	2	3	4	5	6	7	8
`debut_mot(s,i)`	1	1	3	3	4	6	6	8	8

2. À l'aide d'une boucle `while`, écrire une fonction de signature « `fin_mot(s: str, i: int) -> int` » qui renvoie le plus petit indice `j` $\geqslant$ `i` tel que `s[j]` est un espace. Si pour tout `j` $\geqslant$ `i`, le caractère `s[j]` n'est pas un espace, votre fonction renverra `len(s)`. Par exemple, avec `s = " a bc d "`, on obtient :

`i`	0	1	2	3	4	5	6	7	8
`fin_mot(s,i)`	0	2	2	5	5	5	7	7	8

3. À l'aide des fonctions précédentes, écrire la fonction `split`.

Exercice 7. Message de la matrice (exercice facultatif)

Cet exercice est issu du site HackerRank qui propose des exercices de programmation en Python :

`https://www.hackerrank.com/challenges/matrix-script/problem`

La matrice transmet un message complexe à Neo. Le message est une grille rectangulaire composée de caractères alphanumériques et de symboles (`!, @, #, $, %, &`). Un caractère alphanumérique est soit une lettre majuscule (entre `A` et `Z`), soit une lettre minuscule (entre `a` et `z`), soit un chiffre (entre `0` et `9`).

Pour déchiffrer le message, Neo a besoin de lire chaque colonne, de sélectionner les caractères alphanumériques et de les connecter. Neo doit lire chaque colonne du haut vers le bas et commence par la colonne la plus à gauche.

Afin d'améliorer la lisibilité, si le message lu colonne par colonne contient des symboles entre deux caractères alphanumériques, alors Neo les remplace par un unique espace.

Argument de la fonction. La fonction appelée `dechiffrer` prend en argument une chaîne de caractères composée de plusieurs lignes. Toutes les lignes contiennent le même nombre de caractères. Les différentes lignes sont séparées par un retour chariot `"\n"`.

Retour de la fonction. La fonction doit renvoyer le message déchiffré.

Exemple L'exécution du programme ci-contre affiche `"This is Matrix#@&%!"`. En effet, en lisant colonne par colonne, on obtient le message `"This$#is%$Matrix#@&%!"`, puis Neo remplace les symboles entre deux caractères alphanumériques par un simple espace.

```
s = """Tsi
h%x
i$#
sM@
$a&
#t%
ir!"""

print(dechiffrer(s))
```