Exercice 1. Fonctions mystères

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

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

```
def f():
    x = 0
    for i in range(2, 9):
        if i % 3 == 0:
            x += i
        else:
            x -= i
    return x
```

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

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

Exercice 2. Nombres friables

Soient n et B deux entiers supérieurs ou égaux à 2. On dit que n est B-friable si tous les facteurs premiers de n sont inférieurs ou égaux à B. Par exemple, on a $78\,408 = 2^3 3^4 11^2$. Ainsi, 78 408 est 11-friable et 12-friable, mais n'est pas 10-friable.

1. À l'aide d'une boucle `while`, écrire une fonction `suppr_facteur` qui prend en entrée deux entiers n et d, qui divise n par d autant de fois que possible, et renvoie le nombre obtenu lorsqu'on ne peut plus diviser par d. Votre fonction vérifiera que $n > 1$ et $d > 1$ et déclenchera une erreur si ce n'est pas le cas. Par exemple :

`n`	12	1400	78408	7983360	1	78408
`d`	2	3	3	12	3	1
`suppr_facteur(n,d)`	3	1400	968	385	Erreur	Erreur

2. À l'aide d'une boucle `while` et de la fonction précédente, écrire une fonction `get_B` qui prend en entrée un entier n et renvoie le plus petit entier B tel que n est B-friable. Votre fonction vérifiera que $n > 1$ et déclenchera une erreur si ce n'est pas le cas. Par exemple :

`n`	36	1024	78408	417977	65537	1
`get_B(n)`	3	2	11	71	65537	Erreur

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 × 5 × 6 × 3 × 8 = 720` et lorsqu'on multiplie les chiffres de `720` on obtient `7 × 2 × 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 N_k le plus petit entier naturel dont la persistance multiplicative vaut k. Par exemple, $N_0 = 0$ et $N_1 = 10$.

3. (a) Sur feuille, déterminer N_2 et N_3.
   (b) À l'aide d'une boucle `while`, écrire une fonction de signature `get_Nk(k: int) -> int` qui renvoie N_k. Tester votre fonction pour $k \in [\![0 ; 9]\!]$
   (c) **(facultatif, difficile)** Améliorer la fonction précédente pour calculer N_{10} et 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 12/11/2023 à 20h

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

Étant donnée une liste de chaînes de caractères `L` et une chaîne de caractères `sep`, l'évaluation de `sep.join(L)` permet de concaténer les éléments de `L` en les séparant 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	`""`

Écrire une fonction de signature « `join_bis(sep: str, L: list[str])` » telle que que l'appel à `join_bis(sep, L)` s'évalue en la même valeur que `sep.join(L)`. Testez votre fonction sur les exemples ci-dessus.

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

Étant donnée une chaîne de caractères `s`, l'appel à `s.split()` renvoie une liste constituée des mots de `s`. Par exemple :

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

Écrire une fonction de signature « `split_bis(s: str) -> list[str]` » telle que l'appel à `split_bis(s)` s'évalue en la même valeur que `s.split()`. Testez votre fonction sur les exemples ci-dessus. Si besoin, on pourra lire les indications ci-dessous.

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. En déduire la fonction `split_bis`.

Exercice 7. Recherche de facteurs dans une chaîne de caractères

Soient `r` et `s` deux chaînes de caractères. On dit que `r` est un ***facteur*** de `s` s'il existe deux autres chaînes de caractères `u` et `v` telles que `s` soit égale à `u + r + v`. Dans le cas où `u` est la chaîne vide, on dit que `r` est un ***préfixe*** de `s`. Par exemple, les sept préfixes de la chaîne de caractères `"Poinca"` sont :

`""`, `"P"`, `"Po"`, `"Poi"`, `"Poin"`, `"Poinc"`, `"Poinca"`.

Voici des exemples de facteurs pour la chaîne de caractères `"Poinca"` :

`"oi"`, `"n"`, `"nca"` et les sept préfixes donnés ci-dessus.

1. À l'aide d'une boucle `while`, écrire une fonction de signature « `est_prefixe(r: str, s: str) -> bool` » qui renvoie `True` si `r` est un préfixe de `s` et renvoie `False` sinon. Dans cette question, vous ne devez pas utiliser l'instruction `break` ni de sortie anticipée de fonction (pas de `return` dans la boucle `while`). Vérifiez que :

   `est_prefixe("","test")` vaut `True`, `est_prefixe("test","")` vaut `False`,
   `est_prefixe("t","test")` vaut `True`, `est_prefixe("te","test")` vaut `True`,
   `est_prefixe("mot","mot")` vaut `True`, `est_prefixe("mots","mot")` vaut `False`,
   `est_prefixe("ot","mot")` vaut `False`.

2. À l'aide de la fonction `est_prefixe`, écrire une fonction qui, étant donné deux chaînes de caractères `r` et `s`, renvoie `True` si `r` est un facteur de `s` et renvoie `False` sinon. Vérifiez que :

   `est_facteur("abc", "ababc")` vaut `True`, `est_facteur("abc", "abac")` vaut `False`,
   `est_facteur("de", "abcdefg")` vaut `True`, `est_facteur("cab", "abac")` vaut `False`.

Exercice 8. 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))
```