Séquences - Première Approche

1 Chaînes de caractères

Exercice 1. *Autre fonction*

Implémentez une fonction `autre_fonction` qui prend en entrée une chaîne de caractères `s` et un entier naturel `n` et qui renvoie la chaîne de caractères formés de `n` fois `s`, où chaque occurrence de `s` est séparée d'une autre par un `" - "`.
e.g. : `autre_fonction("Nadine",4) = "Nadine - Nadine - Nadine - Nadine"`.

Exercice 2. *Alphabet*

1. Implémentez une fonction `alphabet` qui prend en entrée un caractère `c` minuscule et qui affiche l'alphabet à partir de `c` jusqu'à `'z'`.
2. Améliorez la fonction précédente pour qu'elle affiche l'alphabet en entier auquel on a appliqué une rotation pour commencer à la lettre `c`.
   e.g l'alphabet obtenu après rotation pour commencer à `'j'` donne : `'jklmnopqrstuvwxyzabcdefghi"`

Exercice 3. *Inverser la casse*

Implémentez une fonction `inverser_casse` qui prend en entrée une chaîne de caractères composées de lettres majuscules, minuscules et d'espaces et qui renvoie la même chaîne avec la casse inversée : les majuscules deviennent minuscules, les minuscules, majuscules et les espaces restent inchangées.
Indication : calculer le décalage entre le code ASCII d'une majuscule et de sa version minuscule.

2 Tuples

Exercice 4. *Produit Vectoriel*

On rappelle la formule du produit vectoriel de deux vecteurs $u = (u_1, u_2, u_3)$ et $v = (v_1, v_2, v_3)$ de $\mathbb{R}^3$:

$$u \wedge v = \begin{pmatrix} u_2v_3 - u_3v_2 \\ u_3v_1 - u_1v_3 \\ u_1v_2 - u_2v_1 \end{pmatrix}$$

Implémentez en Python une fonction `produit_vec` qui prend en entrée deux vecteurs de $\mathbb{R}^3$ représentés par des triplés et qui renvoie le résultat de leur produit vectoriel.

Exercice 5. *Triangles*

Implémentez une fonction `triangle` qui prend en entrée trois points de $\mathbb{R}^2$ sous la forme de couples de flottants et qui renvoie trois booléens `R, I, E` sous la forme d'un triplé. `R` sera vrai si le triangle est rectangle, `I` si il est isocèle, `E` s'il est équilatéral.

3 Listes

Exercice 6. ♥ *Somme des éléments d'une liste*

- Simulez pas à pas la fonction `somme_liste` avec pour entrée : `[12,21,24,144]`
- Annotez chaque ligne avec une description de son effet.
- Modifiez la fonction pour calculer le produit des éléments de la liste donnée en entrée.
- Modifiez la fonction pour calculer la sommes des *nombres pairs* de la liste donnée en entrée.

Exercice 7. *Algorithme sur une liste*

Que calcule l'algorithme $\mathcal{M}$?

Algorithme $\mathcal{M}$

ENTRÉES: L une liste d'entiers positifs

```
1.  m ← 0
2.  i ← 0
3.  n ← longueur(L)
4.  tant que i < n faire
5.      si L[i] > m alors
6.          m ← L[i]
7.      fin si
8.      i ← i + 1
9.  fin tant que
10. renvoyer m
```

Exercice 8. ♥ *Maximum, minimum, moyenne, variance*

Implémentez quatre fonctions `imax`, `imin`, `moyenne`, et `variance` qui prennent en argument une liste d'entiers et qui renvoient, respectivement, l'indice du maximum, l'indice du minimum, la moyenne et la variance des valeurs de la liste.

Exercice 9. ♥ *Recherche Linéaire*

Implémentez une fonction `recherche` qui prend en entrée une liste d'entiers l et un entier x et qui renvoie l'indice dans la liste de la première occurrence de x dans l. Si x n'apparaît pas dans l, la fonction devra renvoyer -1.

Exercice 10. *Somme de deux listes*

Implémentez une fonction `somme` qui prend en entrée deux listes de même longueur $L_1 = [a_1, \ldots, a_n]$ et $L_2 = [b_1, \ldots, b_n]$, qui renvoie la liste $L = [a_1 + b_1, \ldots, a_n + b_n]$ obtenue en sommant un à un les éléments de L_1 et L_2.

Exercice 11. *Liste des diviseurs*

Implémentez une fonction `diviseurs` qui prend en entrée un entier strictement positif `n` et qui renvoie une liste contenant les diviseurs de `n`.

4 POUR S'ENTRAÎNER

Exercice 12. *CamelCase*

Le *CamelCase* consiste à écrire une phrase en écrivant chaque mot avec une majuscule **uniquement** en début de mot et en supprimant les espaces. *e.g* la phrase `"Nadine code comme une heroine."` s'écrira `"NadineCodeCommeUneHeroine"`.
Implémentez une fonction `camel_case` qui prend en entrée une chaîne de caractère minuscule, majuscule et d'espaces et qui renvoie une chaîne contenant le même texte en CamelCase.

Exercice 13. *Fake Verlan*

Implémentez une fonction `fake_verlan` qui prend en entrée une chaîne de caractère et renvoie une chaîne de caractère où l'ordre des caractères de la première chaîne est inversé. *e.g* `"Nadine n'a pas besoin de caféine."` devient `".eniéfac ed nioseb sap a'n enidaN"`

Exercice 14. *Produit scalaire*

Implémentez une fonction `produit_scalaire` qui prend en entrée deux vecteurs de $\mathbb{R}^2$, représentés par le couple de leurs coordonnées et qui renvoie le produit scalaire de ces vecteurs.

Exercice 15. *Minimum de liste*

Implémentez une fonction `min` qui prend en entrée une liste d'entier et renvoie le minimum de la liste.

Exercice 16. *Moyenne pondérée*

Implémentez en Python la fonction `moyenne_ponderee` qui prend en entrée une liste de nombres entiers positifs `L` et une liste de coefficient `C` de même longueur et qui renvoie la moyenne de la liste `L` pondérée par les coefficient de C.

Exercice 17. *Éléments impairs*

Implémentez une fonction `impairs` qui prend en entrée une liste d'entiers et qui renvoie une liste contenant les éléments impairs de la liste d'entrée.

Exercice 18. *Liste des multiples*

Implémentez une fonction `multiples` qui prend en entrée deux entiers naturels non nuls n et m et qui renvoie la liste des multiples de `m` plus petits que `n`.

Exercice 19. *Encadrement*

Implémentez en Python la fonction `encadrement` qui prend en entrée une liste de nombres entiers positifs `L` et un nombre entier `x` et qui renvoie le couple `(a,b)` tel que `a` est le plus grand élément de `L` plus petit ou égal à `x` (-1 s'il n'y en a pas) et `b` est le plus petit élément de `L` plus grand que `x` (-1 s'il n'y en a pas).

5 POUR ALLER PLUS LOIN

Exercice 20. *Verlan*

Implémentez une fonction `verlan` qui prend en entrée une chaîne de caractère et renvoie une chaîne de caractère où **pour chaque mot** de la première chaîne, l'ordre des caractères du mot est inversé. *e.g* `"Nadine n'a pas besoin de caféine."` devient `enidaN n'a sap nioseb ed eniéfac.`.
On pourra commencer par considérer que la chaîne en entrée ne contient que des lettres et des espaces.

Exercice 21. *Crible d'Eratostene*

Le crible d'Eratostene est un algorithme permettant d'obtenir la liste des nombres premiers plus petit qu'un entier fixé $n \geq 2$. Le crible fonctionne comme ceci :

1. noter tous les nombres entiers de 2 à n
2. prendre le premier nombre qui n'est pas éliminé.
   — il est premier, le noter dans la liste des nombres premiers.
   — éliminer ce nombre et tous ses multiples de la liste initiale.
3. Répéter `2.` tant que tous les nombres de la liste initiale ne sont pas barrés.

Implémentez une fonction `eratostene` qui prend en entrée un entier `n` supérieur à 2 et qui renvoie la liste des nombres premiers plus petits ou égaux à `n` en utilisant le crible d'Eratostene.

Exercice 22. *Calcul de la Médiane*

Implémentez deux fonctions `mediane` qui calcule la médiane d'une liste d'entiers.
Si la liste possède un nombre pair d'éléments, la médiane sera le milieu des deux éléments centraux de la liste. *e.g.* la médiane de `[4,3,8,7,5,7]` est le milieu de `5` et `7` soit `6`.

Exercice 23. *Tri de liste*

Décrire un algorithme en pseudo-code qui permet de renvoyer une copie triée par ordre croissant d'une liste d'entier.
Vérifiez votre algorithme en le simulant sur des exemples pertinents.
Implémentez l'algorithme en python et réalisez un jeu de test pertinent.

Syntaxe Python

Objets Python

Type	Syntaxe	Commentaire
Chaînes de caractère	`"Nadine"`, `'Nadine'`, `"""Nadine"""`, `""`	
Tuples	`(1,2,3)`, `()`	
Listes	`[1,2,3]`, `[]`	

Conversions

```
# Vers Tuple
tuple("abc"), tuple([1,2,3])
# Vers Liste
list("abc"), list((1,2,3))
```

Opérations

Type	Syntaxe
Chaînes de caractère	`s1 + s2`, `s1 * 12`
Tuples	`t1 + t2`, `t1 * 12`
Listes	`l1 + l2`, `l1 * 12`

Comparaisons

Inégalités

`x < y, x > y, x <= y, x >= y`

Compare les chaînes de caractères, les tuples, les listes pour l'ordre lexicographique.

Égalité entre objets

`x == y, x != y`

Chaînes de caractère

Table ASCII

	0	1	2	3	4	5	6	7	8	9
0										
10										
20										
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{		}	~			

Codage de caractère

```
ord('A'), chr(75)
```

```
Sortie : 65 K
```


Programmes du TP

```
def somme_liste(l : list[int]) -> int :
    """
    Entrées : une liste d'entiers relatifs
    Sortie : la somme des entiers de la liste
    """
    i = 0
    n = len(l)
    somme = 0
    while(i < n) :
        somme = somme + l[i]
        i = i + 1
    return somme
```