

TD-TP n°4

Cette feuille d'exercices suit le cours 4 et reprend aussi des notions des cours précédents.

Selon les indications données, les exercices sont à traiter

- en mode TD (papier, crayon),
- en mode TP (Thonny, jupyter-notebook)

Exercice 1. (TD) Parcours d'une chaîne

Proposez plusieurs façons d'afficher, un à un, tous les caractères composant une chaîne donnée. Vous utiliserez pour cela une boucle **for** et vous complèterez le programme suivant dans chacun des deux cas.

```
1 mot = "python"
2
3 for ...
```

Exercice 2. (TD-TP) Analyse de code

Quel est l'affichage produit par le code suivant ? Vous devez d'abord proposer une solution que vous écrivez, puis vous recopiez le code. Si votre solution n'est pas bonne, vous devez comprendre votre erreur.

```
1 for i in range(2, 11, 3) :
2     for j in range(3, 18, 4) :
3         if i + j > 13 :
4             print("A", end = " ")
5         else :
6             print("B", end = " ")
```

Exercice 3. (TD-TP) Analyse de code 2

Quel est l'affichage produit par le code suivant ? Vous devez d'abord proposer une solution que vous écrivez, puis vous recopiez le code. Si votre solution n'est pas bonne, vous devez comprendre votre erreur.

```
1 liste1 = []
2 liste2 = [4, 0, 5, 2, 1, 3]
3
4 for i in range(len(liste2)) :
5     j = liste2[-1-i]
6     n = liste2[j]
7     liste1 = liste1 + [n]
8
9 print(liste1)
```

Exercice 4. (TP) Somme de 1 à n , Prooduit de 1 à n

Écrivez un programme qui a le comportement suivant :

- il demande à l'utilisateur d'entrer un entier n supérieur ou égal à 2. La demande est renouvelée **tant que le nombre donné est inférieur à 2** (on suppose que l'utilisateur donne bien un nombre),
- une fois la demande satisfaite, il calcule la somme et le produit des entiers entre 1 et n , au moyen d'une boucle for,
- enfin, il affiche ces deux résultats.

```
entrez un entier n supérieur ou égal à 2 : 1
entrez un entier n supérieur ou égal à 2 : 0
entrez un entier n supérieur ou égal à 2 : 6

calcul lancé pour n = 6 ...

pour n = 6 :
- s = 21
- p = 720
```

Exercice 5. (TP) Nombres premiers

Un entier naturel est premier s'il a exactement deux diviseurs entiers. De manière équivalente, il est premier s'il est strictement plus grand que 1 et n'est divisible par aucun nombre entre 1 et lui-même exclus.

Écrivez un programme qui a le comportement suivant :

- il demande à l'utilisateur d'entrer un entier n supérieur ou égal à 2. La demande est renouvelée **tant que le nombre donné est inférieur à 2** (on suppose que l'utilisateur donne bien un nombre),
- une fois la demande satisfaite, il teste par des divisibilités successives si le nombre est premier,
- enfin, il affiche le résultat.

Voici des exemples d'exécution :

```
entrez un entier n supérieur à 1 : 1
entrez un entier n supérieur à 1 : 0
entrez un entier n supérieur à 1 : 127
127 est premier
```

```
entrez un entier n supérieur à 1 : 221
221 n'est pas premier, il est divisible par 13
```

Exercice 6. (TD-TP) Nombres premiers v2

Voici un programme. Expliquez ce qu'il fait. Vous détaillerez chaque partie du programme.

```
1 while True :
2     n = int(input("entrez un entier n supérieur à 1 : "))
3     if n >= 2 :
4         break
5
6 nbDiv = 0
7
8 for d in range(2, n) :
9     if n % d == 0 :
10        nbDiv = nbDiv + 1
11
12 print("pour n =", n, ", nbDiv = ", nbDiv)
```

1. Expliquez ce qu'il fait. Vous détaillerez chaque partie du programme.
2. Modifiez le programme pour qu'il détecte :
 - les nombres premiers,
 - les nombres qui sont le carré d'un autre nombre (par exemple 64 ou 121). Pour cela vous pourrez regarder à la main ce qui se passe pour de tels nombres.

Exercice 7. (TP) distance de Hamming

La distance de **Hamming** entre deux chaînes de caractères **non vides et de même longueur** est le nombre d'indices où les deux chaînes n'ont pas le même caractère.

Par exemple :

- entre abcde et acdbe, la distance est 3,
- entre abcde et abcdf, la distance est 1,
- entre abcde et ehgjt, la distance est 5,
- entre abcde et abcde, la distance est 0,

Écrivez un programme qui :

- a) demande une première chaîne de caractères à l'utilisateur, et renouvelle sa demande jusqu'à obtenir une chaîne non vide,
- b) demande ensuite une deuxième chaîne, et renouvelle sa demande jusqu'à obtenir une chaîne de même longueur que la première,
- c) une fois les deux chaînes obtenues, calcule et affiche la distance de Hamming entre les deux chaînes.

Le visuel doit être conforme à :

```
entrez la chaîne de caractères 1 : abcd
la chaîne 1 est abcd. Entrez la chaîne de caractères 2 : abcde
la chaîne 1 est abcd. Entrez la chaîne de caractères 2 : abce
distance entre abcd et abce : 1
```

Exercice 8. (TP) tuples croissants

Au départ, la variable `t` référence un tuple de valeurs comparables. Par exemple, un tuple de nombres, ou bien encore un tuple de chaînes de caractères.

Rappel : l'ordre "naturel" entre chaîne de caractères est l'ordre lexicographique (l'ordre du dictionnaire pour être clair).

L'objectif de l'exercice est de dire si oui ou non le tuple testé est croissant.

Par exemple, `(1, 4, 5, 5, 7)` est croissant, ainsi que `("abc", "bd", "bfc", "ghi", "wzae")`, alors que `(1, 5, 4, 6, 7)` ne l'est pas.

Écrivez un programme qui part d'un tuple donné (vous pourrez changer la valeur du tuple pour les différents tests) et qui affiche "croissant" si les valeurs du tuple sont croissantes et "non croissant" sinon (dans ce cas des détails prouvant la non croissances seront affichés).

On considère que le tuple vide est croissant.

Exemple de rendu :

```
- tuple testé : (1, 4, 4, 5, 8, 9)
(1, 4, 4, 5, 8, 9) est croissant

- tuple testé : (1, 3, 5, 7, 6, 8, 9, 12)
(1, 3, 5, 7, 6, 8, 9, 12) n'est pas croissant car 7 > 6

- tuple testé : ('abc', 'bd', 'cdf', 'be', 'rt')
('abc', 'bd', 'cdf', 'be', 'rt') n'est pas croissant car cdf > be

- tuple testé : ('a', 'bb', 'cccc', 'd')
('a', 'bb', 'cccc', 'd') est croissant

- tuple testé : (1,)
(1,) est croissant

- tuple testé : ()
() est croissant
```

Vous pourrez partir de ce début de code :

```
1 # tuples de tests
2 t1 = (1, 4, 4, 5, 8, 9)
3 t2 = (1, 3, 5, 7, 6, 8, 9, 12)
4 t3 = ("abc", "bd", "cdf", "be", "rt")
5 t4 = ("a", "bb", "cccc", "d")
6 t5 = (1,)
7 t6 = ()
8
9 for t in [t1, t2, t3, t4, t5, t6] :
10     print("- tuple testé : ", t)
11     # code à compléter à partir d'ici
```

Exercice 9. (TP) études de températures

Cet exercice va vous demander de l'autonomie, et probablement du travail d'équipe.

Vous disposez sur Celene d'un fichier **ex9.py**.

Ce fichier présente un tuple nommé **temperatures**, qui est assez difficile à lire sans explication...

Explications :

- le tuple **temperatures** est composé de 30 "sous-tuples",
- chaque sous-tuple correspond à un jour de juin 2025. Ainsi, le sous-tuple (15.3, 26.7, 19.1) est celui du 1er juin, le sous-tuple (12.9, 22.4, 16.6) correspond au 2 juin, etc,
- Pour chaque sous-tuple, la première valeur est la température minimale du jour. La deuxième est la température maximale du jour, et la troisième est la température moyenne du jour.

Vous devez, en écrivant les lignes de code nécessaires, faire afficher les statistiques suivantes :

1. la température minimale observée en juin 2025,
2. de même, la température maximale,
3. la plus forte amplitude journalière (écart entre le min et le max dans une journée),
4. la température moyenne du mois (on pourra estimer que c'est la moyenne des températures moyennes),
5. le nombre de jours où la température minimale a dépassé un seuil fixé à l'avance.