Exercices de révision

Exercices à faire en autonomie sur un ordinateur ou sur feuilles.

Par messages Ecole Directe ou au début/à la fin d'un cours, vous pouvez demander à votre professeur une aide ou une correction des exercices (c'est valable aussi pour les exercices que vous trouvez pas vous-même).

1. On souhaite programmer ses radiateurs en mode « Confort » de 9h à 22h et en mode « Eco » le reste du temps.

On supposera que minuit correspond à l'heure zéro.

La fonction `radiateur(heure)` qui prend en paramètre un `float` représentant une heure de la journée et qui retourne le mode associé à cette heure.

Compléter les « … » du programme suivant :

```
def radiateur(heure):
    if ... :
        return "Heure invalide"

    if heure < 9:
        mode = "Eco"
    elif heure < 22:
        mode = "Confort"
    else:
        mode = ...
    return mode


# Doit afficher uniquement des 'True'
print(radiateur(3) == "Eco")
print(radiateur(9) == "Confort")
print(radiateur(23.2) == ...)
print(radiateur(7.5) == ...)
print(radiateur(12) == ...)
print(radiateur(22) == ...)
print(radiateur(...) == "Heure invalide")
print(radiateur(...) == "Heure invalide")
```

2. Observer la fonction suivante et en déduire ce qu'elle fait.

```
def mystere_2(n:int): # n être un entier positif
    liste = []
    i = 1
    while len(liste) != n:
        if i%5 == 0:
            liste.append(i)
        i+=1
    return liste
```

3. Observer la fonction suivante et en déduire ce qu'elle fait.

```
def mystere_3(liste:list): # liste est de type list
    """ La liste contient uniquement des entiers """

    for i in range(1, len(liste)):
        if liste[i-1] > liste[i]:
            return False
    return True
```

4. Ecrire une fonction `compter(L)` qui prend en paramètre une liste d'entiers. La fonction calcule la moyenne des entiers et retourne un tuple de deux valeurs : le nombre d'entiers strictement inférieurs à la moyenne et le nombre d'entiers strictement supérieurs à la moyenne.

Le programme doit afficher `True` pour tous les `print` suivants :

```
print(compter([5]) == (0, 0))
print(compter([1, 2, 4, 5, 6]) == (2, 3))
print(compter([10, 2, 10, 1, 15, 11, 1, 12, 13]) == (3, 6))
print(compter([2, 4, 6, 8, 10, 12]) == (3, 3))
print(compter([2, 4, 6, 8, 10, 12, 14]) == (3, 3))
```

5. La fonction `extrait` prend en paramètres une liste `L` et deux entiers `i` et `j` et retourne une liste composée dans éléments de `L` de son indice `i` à son indice `j` inclus.

a. Ecrire cette fonction afin que le programme suivant affiche uniquement `True` :

```
li = [9, 5, 1, "r", "t", "y", 7.3, True, (2, 5), 8, (9, 4)]
print(extrait(li, 2, 4) == [1, "r", "t"])
print(extrait(li, 0, 6) == [9, 5, 1, "r", "t", "y", 7.3])
print(extrait(li, 8, 10) == [(2, 5), 8, (9, 4)])
print(extrait(li, 7, 8) == [True, (2, 5)])
print(extrait(li, 7, 7) == [True])
```

b. Ta fonction marche-t-elle avec d'autres types de structures qu'une liste (dictionnaire, tuple ou chaine de caractères) ?

Bonus : Proposer une autre version de cette fonction

6. Compléter le programme suivant afin qu'il affiche `True` :

```
def tri_par_matieres(notes):
    dico_matieres = {}
    for tpl in notes.values():
        matiere = ...
        if matiere in dico_matieres.keys():
            dico_matieres[matiere].append(...)
        else:
            ...
    return dico_matieres

notes = {"Emma":("Math", 16), "Lucas":("NSI", 14),
         "Manon":("NSI", 13), "Dylan":("SES", 18),
         "Morgan":("Math", 9)}

print(tri_par_matieres(notes) == {"Math":[16, 9],
                                  "NSI": [14, 13],
                                  "SES": [18]})
```

7. Observer la fonction suivante et en déduire ce qu'elle fait.

```
def mystere_7(n): # n être un entier positif
    return [i for i in range(n)]
```

8. Observer la fonction suivante et en déduire ce qu'elle fait.

```
def mystere_8(n): # n être un entier positif
    return [[ j for j in range(i+1)] for i in range(n)]
```

9. Observer la fonction suivante et en déduire ce qu'elle fait.

```
def mystere_9(n): # n être un entier positif
    return [i if i%2==0 else i*10 for i in range(n)]
```

10. La fonction `minimum` prend en paramètres une liste d'entiers `entiers` et retourne d'indice du plus petit élément de la liste.

Ecrire cette fonction afin que le programme suivant affiche uniquement `True` :

```
print(minimum([9, 5, 1, 3, 0, 7, 6, 4, 8, 2]) == 4)
print(minimum([9, 5, 1, 3, 7, 6, 4, 0, 8, 2]) == 7)
print(minimum([0, 9, 5, 1, 3, 7, 6, 4, 8, 2]) == 0)
print(minimum([9, 5, 1, 3, 7, 6, 4, 8, 2, 0]) == 9)
```

11. La fonction `top_trois` prend en paramètres une liste d'entiers `entiers` et retourne un tuple avec les indices des trois plus petits éléments de la liste.

Ecrire cette fonction afin que le programme suivant affiche uniquement `True` :

```
print(top_trois([]) == (None, None, None))
print(top_trois([9]) == (0, None, None))
print(top_trois([9, 4]) == (1, 0, None))
print(top_trois([9, 0, 4]) == (1, 2, 0))
print(top_trois([1, 9, 5, 3, 0, 7, 6, 4, 2]) == (4, 0, 8))
print(top_trois([2, 9, 5, 1, 3, 7, 6, 0, 4]) == (7, 3, 0))
print(top_trois([0, 9, 5, 3, 2, 7, 6, 4, 1]) == (0, 8, 4))
print(top_trois([9, 5, 1, 3, 7, 6, 4, 2, 0]) == (8, 2, 7))
```

12. Au Scrabble, les joueurs/joueuses gagnent des points en fonction :

- De la longueur du mot : 50 points si les 7 lettres sont posées en une fois.
- Des lettres du mot : A vaut 1 points, B vaut 3, W vaut 10, etc.
- Des cases sur lesquelles les lettres sont posées (on ne va pas les prendre en compte pour cet exercice).

On dispose du dictionnaire `scrabble` qui associe chaque lettre à un nombre de points :

```
scrabble = {"A":1, "B":3, "C":3, "D":2, "E":1, "F":4, "G":2,
            "H":4, "I":1, "J":8, "K":5, "L":1, "M":2, "N":1,
            "O":1, "P":3, "Q":8, "R":1, "S":1, "T":1, "U":1,
            "V":4, "W":10, "X":10, "Y":10, "Z":10}
```

a. Ecrire la fonction `compter_points` qui prend en paramètres `mot` une chaine de caractères non-vide d'au plus 7 lettres et `scrabble` un dictionnaire, et qui retourne le nombre de points du mot.

Le programme suivant doit afficher uniquement `True` :

```
print(compter_points("AEIOU", scrabble) == 5)
print(compter_points("XYZ", scrabble) == 30)
print(compter_points("AVION", scrabble) == 8)
print(compter_points("PARADER", scrabble) == 60)
```

b. Ajouter une condition pour vérifier si le paramètre `mot` correspond aux attentes des consignes. S'il ne correspond pas, la fonction retourne `None`.

13. Certains animaux de compagnie sont nommés en fonction de leur année de naissance. Par exemple, un animal né en 2025 aura un prénom commençant par « A », en 2026 par un « B », en 2027 par un « C », etc.

Depuis 1985 (année en « A »), on ne prend plus en compte les lettres K, Q, W, X, Y et Z.

Compléter le programme suivant pour faire en sorte que la fonction `calculer_nom(annee)` retourne la première lettre du prénom d'un animal né cette `annee`-là.

```
def calculer_nom(annee):

    alphabet = "ABCDEFGHIJLMNOPRSTUV"
    annee_debut = 1985  # lettre A
    pass

# Doit afficher uniquement des 'True'
print(calculer_nom(1985) == "A")
print(calculer_nom(1988) == "D")
print(calculer_nom(1997) == "N")
print(calculer_nom(2025) == "A")
print(calculer_nom(2026) == "B")
```

14. La suite de Fibonacci est une suite de nombres entiers dont chaque terme successif représente la somme des deux termes précédents, et qui commence par 0 puis 1.

On note généralement les deux premières valeurs $U_0 = 0$ et $U_1 = 1$ et la suite est notée $U_n = U_{n-1} + U_{n-2}$ (avec $n \geq 2$).

Ainsi :

$U_2 = U_1 + U_0 = 1$ $\qquad$ $U_4 = U_3 + U_2 = 3$

$U_3 = U_2 + U_1 = 2$ $\qquad$ $U_5 = U_4 + U_3 = 5$

a. Calculer U_8.

b. Ecrire une fonction `u(n)` qui retourne la valeur de la suite de Fibonacci pour U_n.

c. Vérifier le bon fonctionnement de la fonction pour chaque valeur de U_0 à U_8.

15. On dispose d'un dictionnaire de mots en français et en allemand.

```
mots = { "hiver":"fr", "nordique":"fr", "winter":"al",
         "loup":"fr",  "schloss":"al",  "koniginnen":"al",
         "reines":"fr", "nordisch":"al"}
```

On souhaite réaliser un programme qui permet de classer automatiquement un mot dans l'une de ces langues en le comparant aux mots déjà connus.

a. De quel type d'algorithme a-t-on besoin pour réaliser un tel programme ?

On décide de classer ces mots en fonctions de leur longueur et de leur nombre de voyelles (a, e, i, o, u et y).

b. Ecrire une fonction `nb_voyelle(mot)` qui prend en paramètre un mot de type `str` et retourne son nombre de voyelles. Le programme doit afficher `True` :

```
print(nb_voyelle("rtpdf") == 0)
print(nb_voyelle("azerty") == 3)
print(nb_voyelle("abababa") == 4)
print(nb_voyelle("aeyuio") == 6)
```

On suppose que la distance entre deux mots m1 et m2 est calculée avec cette formule

$$\sqrt{(Lm1 - Lm2)^2 + (Vm1 - Vm2)^2}$$

sachant que Lm1 et Lm2 sont respectivement les longueurs de m1 et m2 et Vm1 et Vm2 le nombre de voyelles dans les deux mots précédents.

On rappelle que la racine carrée d'un nombre `n` peut s'écrire $n^{1/2}$ **.**

c. Ecrire une fonction `calcul_distance` qui prend en paramètres deux mots et qui retourne la distance entre les deux mots. Le programme doit afficher `True` :

```
print(calcul_distance("abc", "abc") == 0)
print(calcul_distance("abcdef", "age") == 3)
```

d. La fonction `tri(dico_mots, mot_mystere)` prend en paramètre un dictionnaire de mots, dont les clés sont les mots et les valeurs sont les langues des mots et un mot mystère que l'on veut classer.

La fonction retourne une liste des mots classés par ordre croissant de distances avec le mot mystère.

```
def tri(dico_mots, mot_mystere):
    li, dist = [], []
    for mot in dico_mots. ... ():
        li.append(mot)
        dist.append(calcul_distance(...))
        i = len(li)-1
        while i > 0 and dist[...] > dist[...]:
            ... # Inverser 2 éléments de la liste li
            ... # Inverser 2 éléments de la liste dist
            i-=1
    return li
```

- Quel algorithme de tri est effectué dans la fonction précédente ?
- Compléter cette fonction afin que le programme affiche `True` :

```
liste = ['nordique', 'koniginnen', 'nordisch',
    'schloss', 'reines', 'winter', 'hiver', 'loup']
print(tri(mots, "kaninchen") == liste)
```

e. La fonction `proches_voisins (dico_mots, mot_mystere)` prend en paramètre un dictionnaire de mots et un mot mystère.

La fonction retourne un dictionnaire des mots les plus proches du mot recherché. Tout comme `dico_mots`, les clés du dictionnaire retourné seront des mots et les valeurs associées seront les langues.

```
def proches_voisins(dico_mots, mot_mystere):
    voisins = {}
    mots_tries = ...
    for i in range(3):
        mot = mots_tries[...]
        voisins[mot] = ...
    return voisins
```

- Combien de mots seront retournés par cette fonction ?
- Compléter cette fonction afin que le programme affiche `True` :

```
dicoA = {'hiver': 'fr', 'winter': 'al', 'loup': 'fr'}
print(proches_voisins(mots, "lapin") == dicoA)

dicoB = {'nordique': 'fr', 'koniginnen': 'al',
         'nordisch': 'al'}
print(proches_voisins(mots, "kaninchen") == dicoB)
```

f. Pour finir, écrire la fonction `majoritaire` qui prend en paramètre un dictionnaire de mots et qui retourne `"fr"` si le dictionnaire contient en majorité des mots français et `"al"` sinon.

Le programme suivant doit afficher uniquement `True` :

```
print(majoritaire(dicoA) == "fr")
print(majoritaire(dicoB) == "al")
```