Exercice 1.

Définissez une fonction maximum(n1,n2,n3) qui renvoie le plus grand des 3 nombres n1, n2, n3 fournis en arguments. Par exemple, l'exécution de l'instruction : print(maximum(2,5,4)) doit donner le résultat :5.

Exercice 2.

Définissez une fonction changeCar(ch,ca1,ca2,début,fin) qui remplace tous les caractères ca1 par des caractères ca2 dans la chaîne de caractères ch, à partir de l'indice début et jusqu'à l'indice fin, ces deux derniers arguments pouvant être omis (et dans ce cas, la chaîne est traitée d'une extrémité à l'autre). Exemples de la fonctionnalité attendue :

Exercice 3.

Un gardien de phare va aux toilettes cinq fois par jour. Or les WC sont au rez-de-chaussée… Écrire une procédure (donc sans return) **hauteurParcourue** qui reçoit deux paramètres, le nombre de marches du phare et la hauteur de chaque marche (en cm), et qui affiche :

Pour x marches de y cm, il parcourt z.zz m par semaine.

On n'oubliera pas :
qu'une semaine comporte 7 jours ;
qu'une fois en bas, le gardien doit remonter ;
que le résultat est à exprimer en m.

Exercice 4.

Un permis de chasse à points remplace désormais le permis de chasse traditionnel. Chaque chasseur possède au départ un capital de 100 points. S'il tue une poule, il perd 1 point, 3 points pour un chien, 5 points pour une vache et 10 points pour un ami. Le permis coûte 200 euros.

Écrire une fonction amende qui reçoit le nombre de victimes du chasseur et qui renvoie la somme due. Utilisez cette fonction dans un programme principal qui saisit le nombre de victimes et qui affiche la somme que le chasseur doit débourser.

Exercice 5.

La suite (ou conjecture) de Syracuse est une suite assez originale : comme les suites classiques, chaque terme est issu du précédent. L'originalité réside dans le calcul des termes... Si le terme en cours "n" est pair, alors le terme suivant est "n/2". Et s'il est impair alors le terme suivant est "3*n+1" (une variante possible nommée "suite compressée" part du principe que 3*n+1 étant forcément pair le terme suivant sera alors n/2 ce qui autorise alors à sauter le n/2 en calculant directement (3*n+1)/2).Son comportement à l'avance est incalculable (il est impossible de donner le terme "n+1" sans avoir calculé les "n" précédents). La conjecture dit que, quel que soit le nombre "n" initial (mais au moins positif), on arrivera fatalement à une finale 4, 2, 1 (ou 2, 1 pour la version compressée). Et de là on ne peut plus en sortir (puisque "1" est impair,

le terme qui le suit est donc 3*n+1 soit 4 et l'on voit alors la boucle inévitable (4, 2, 1, 4, 2, 1, 4, 2, 1, ...).Mais avant d'arriver à cette finale 4, 2, 1, les nombres de la suite varient de façon quasi chaotique. Ce comportement erratique assimilable à celui d'une feuille tombant dans le vent a donné naissance à un vocabulaire issu de l'aéronautique. On nomme par exemple "durée de vol" le nombre de termes calculables à partir d'une valeur de départ. Et l'"altitude" est la valeur maximale atteinte durant le calcul. Par exemple si on part de 14, la suite donnera 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1 soit une durée de vol de 18 et une altitude de 52.

L'exercice proposé est d'écrire une fonction qui, à partir d'un nombre "n", calcule tous les termes de la suite issue de ce nombre (elle s'arrête quand elle atteint 1). Et elle devra renvoyer finalement sous forme d'un tuple

- le nombre initial
- la durée de vol
- l'altitude

En supplément facultatif, on pourra utiliser la fonction et son retour pour tenter de trouver un nombre "X" ayant la durée la plus longue et un nombre "Y" donnant l'altitude la plus haute (X et Y cherchés par exemple dans le premier million d'entiers) ...

Exercice 6.

Soit des comptes bancaires d'individus définis par la liste :

```
Comptes = [     {'nom': 'Boismoneau', 'prenom': 'stephane', 'epargne': 2500},
                {'nom': 'Jambon', 'prenom': 'fred', 'epargne': 5000},
                {'nom': 'Durois', 'prenom': 'nicolas', 'epargne': 10000},
                {'nom': 'Gueux', 'prenom': 'phillipe', 'epargne': 1250},
                {'nom': 'Duchan', 'prenom': 'alice', 'epargne': 4530},
                {'nom': 'Lepenou', 'prenom': 'amed', 'epargne': 2200},
                {'nom': 'Gueux', 'prenom': 'bernard'},
                {'nom': 'Jambon', 'prenom': 'steven', 'epargne': 1670},
                {'nom': 'Gueux', 'prenom': 'sylvie', 'epargne': 3},
                {'nom': 'Durois', 'prenom': 'berbard', 'epargne': 300000}
          ]
```

On considère que les individus qui portent le même 'nom' sont de la même famille. En cas d'absence de revenu attribué à un individu, nous considérerons que son épargne est nulle (cas de 'bernard Gueux').

Écrire une fonction qui retourne le nom de la famille à la plus faible épargne ainsi que le nom de la famille à la plus forte épargne avec le montant de leur épargne respective. Ici, ('Gueux', 1253) et ('Durois', 310000).

Exercice 7

Dans la série "Kaamelott" d'Alexandre Astier, le "cul de chouette" est le jeu favori du tavernier (Alain Chapuis) et du chevalier Karadoc (Jean-Christophe Hembert). Le but présumé de ce jeu (qui n'est pas vraiment expliqué dans la série) est de jeter des dés en tentant d'atteindre un certain total par jet.

Le but de cet exercice est d'écrire une fonction qui, pour une valeur donnée, renvoie toutes les solutions de 3 dés (dés classiques allant de 1 à 6) pouvant donner cette valeur. Attention, les solutions doivent être uniques. Si la solution (1, 2, 3) convient pour la valeur 6 alors la solution (2, 3, 1) ne peut plus convenir (les dés sont interchangeables).

Aide: renvoyer plusieurs valeurs consiste à créer un tableau et à le remplir durant la recherche puis au final retourner le tableau.

Exercice 8

Soit deux listes n1 et n2 permettant de coder deux entiers naturels. Par exemple :

- n1 = [9, 4, 0] pour coder le nombre 940 ;
- n2 = [8, 3] pour coder le nombre 83(*).

(*) On s'interdit les nombres commençant par 0 (à l'exception du chiffre 0). Ainsi, le nombre 83 sera codé [8, 3], et non [0, 8, 3].

Le but est de faire la somme n1+n2 et de retourner le résultat dans le même format de liste, soit: somme = [1, 0, 2, 3] car 940 + 83 = 1023

```
>>> somme_2nombres([9, 4, 0], [8, 3])
>>> [1, 0, 2, 3]

>>> somme_2nombres([1, 9, 3], [7]))
>>> [2, 0, 0]

>>> somme_2nombres([1, 2, 3], [0])
>>> [1, 2, 3]

>>> somme_2nombres([0], [0])
>>> [0]
```

Exercice9

Soit une liste d'entiers, il s'agit d'écrire une fonction qui renvoie les couples des indices d'éléments de la liste, de telle sorte que la somme de ces deux éléments est égale à la valeur cible choisie.

Par exemple :

Code python :

```
>>> somme_2([-2, -1, 2, 1], cible = 0)
>>> [(0, 2), (1, 3)]
```

car les deux nombres aux indices 0 et 2, c-à-d respectivement -2 et 2, donnent la valeur cible = 0 si on les additionne (-2 + 2 = 0).

De même pour le couple (1, 3) où les éléments correspondants aux indices 1 et 3, c.-à-d. respectivement -1 et 1, donnent aussi la valeur cible lorsqu'on les additionne (-1 + 1 = 0).

1. Si la cible ne peut être atteinte avec les entrées, la fonction renvoie une liste vide : []
2. Les indices renvoyés dans un couple doivent être différents : (i, j) tels que i ≠ j.
3. Pour un couple d'indices donné, vous pouvez renvoyer indifféremment (i, j) ou (j, i), mais pas les deux.
4. De même, l'ordre des couples dans la liste est aussi indifférent.

```
>>> somme_2([2, 9, 5, 3, -1], cible = 6)
>>> []

>>> somme_2([-2, -2, -1, -1, 1, 2, 2], cible = 0)
>>> [(0, 5), (0, 6), (1, 5), (1, 6), (2, 4), (3, 4)])
```

Exercice 10

Voici un schéma de la pyramide en question :

Code

```
   1
  2 3
 6 4 5
8 1 2 9
```

En partant du sommet de la pyramide (1), vous pouvez à chaque étape descendre d'un étage sur un nombre adjacent.

Du sommet (1), vous pouvez vous rendre sur (2) ou sur (3) juste en dessous. Si vous êtes descendu sur (2), vous pouvez poursuivre sur (6) ou sur (4). Mais si vous aviez choisi plutôt de descendre sur (3), vous pouvez poursuivre sur (4) ou sur (5), etc.

Si on modélise la pyramide dans une liste de listes :

Code python :

```
pyramide = [ [1], [2,3], [6,4,5], [8,1,2,9] ]
```

Et si vous êtes situé à la position pyramide[i][j], vous pouvez donc vous déplacer soit sur pyramide[i+1, j], soit sur pyramide[i+1, j+1].

Le chemin vers la sortie en bas de la pyramide est celui qui minimisera la somme des nombres que vous rencontrerez au passage.

Par exemple, ici le chemin sera (nombres soulignés):
Code :

```
   1
  2 3
 6 4 5
8 1 2 9
```

car 1 + 2 + 4 + 1 = 8 est le chemin qui minimise la somme des nombres rencontrés parmi tous les chemins possibles entre le sommet et la base de la pyramide.

Vous devez écrire une fonction meilleur_chemin() qui renvoie cette somme :
Code python :

```
>>> pyramide = [ [1], [2,3], [6,4,5], [8,1,2,9] ]
>>> meilleur_chemin(pyramide)
>>> 8
```

Code python :

```
>>> pyramide = [ [1], [2, 3], [6, 4, 5], [8, 1, 2, 9], [7, 0, 2, 1, 3] ]
>>> meilleur_chemin(pyramide)
>>> 8
```

Code python :

```
>>> pyramide = [ [42] ]
>>> meilleur_chemin(pyramide)
>>> 42
```

Code python :

```
>>> pyramide = [ [1], [5, 4], [1, 4, 3], [3, 4, 2, 1], [1, 2, 5, 5, 5] ]
>>> meilleur_chemin(pyramide)
>>> 11
```

Exercice 11

Ecrire un programme Python qui définit une fonction *Positif* qui prend un entier comme argument, et revoie *True* s'il est positif et *False* sinon. Définir ensuite la fonction *AfficherPositifs* qui prend comme argument une liste et affiche ses nombres positifs en exploitant la fonction *Positif*. Lire ensuite une liste d'entiers et afficher ses nombres positifs en faisant appel à la fonction *AfficherPositifs*.

Exercice 12

Ecrire un programme Python qui définit une fonction *Lire* qui permet de lire une liste d'entiers. Puis utiliser cette fonction pour lire deux listes. Assembler ensuite les deux listes dans une troisième et afficher les trois.

Exercice 13

Ecrire un programme Python qui définit une fonction *Conversion* qui prend comme argument un nombre entier et l'affiche dans les trois bases ; binaire, octale et hexadécimale, en exploitant les fonctions de conversion *bin*, *oct* et *hex*.

Utiliser cette fonction pour afficher les équivalents binaire, octal et hexadécimal d'un entier entré par l'utilisateur.

13 Exercices Sur les fonctions en py