Devoir surveillé d'informatique

A Consignes

- On pourra toujours librement utiliser une fonction demandée à une question précédente même si cette question n'a pas été traitée.
- Veillez à présenter vos idées et vos réponses partielles même si vous ne trouvez pas la solution complète à une question.
- La clarté et la lisibilité de la rédaction et des programmes sont des éléments de notation.

☐ Exercice 1 : Question de cours

On considère la fonction puissance ci-dessous :

1. Montrer que cette fonction est correcte en mettant en évidence un invariant de boucle (sa spécification est donnée en commentaire dans le code ci-dessus).

Montrons que la propriété « P(k) : après k tours de boucles, la variable p contient x^k » est un invariant de boucle.

- P(0) est vrai puisqu'avant d'entrer dans la boucle $p=0=x^0$.
- En supposant P(k) vraie, $p==x^k$ et donc au tour de boucle suivant $p=x^k*x$, donc $p=x^{k+1}$.

Donc P(k) est vrai pour tout $k \in \mathbb{N}$. Comme on effectue n tour de boucles, à la sortie de la boucle $p = x^n$ et donc la fonction est correcte.

2. Ecrire une version récursive de cette fonction et prouver qu'elle termine.

```
def puissance_rec(x,n):
    assert type(n)==int and n>=0
    assert type(x)==int or type(x)==float
    if n==0:
        return 1
    return x*puissance_rec(x,n-1)
```

Cette fonction termine car ${\tt n}$ est un variant, en effet :

- n est à valeurs dans \mathbb{N} , puisque $n \ge 0$ par précondition.
- n décroît strictement, en effet n diminue de 1 à chaque appel récursif.

☐ Exercice 2 : anagrammes

Deux mots de $m\hat{e}me$ longueur sont anagrammes l'un de l'autre lorsque l'un est formé en réarrangeant les lettres de l'autre. Par exemple :

- *niche* et *chien* sont des anagrammes.
- epele et pelle, ne sont pas des anagrammes, en effet bien qu'ils soient formés avec les mêmes lettres, la lettre l ne figure qu'à un seul exemplaire dans epele et il en faut deux pour écrire pelle.

Le but de l'exercice est d'écrire une fonction anagrammes qui prend en argument deux chaines de caractères et qui renvoie true si ces deux chaines sont des anagrammes et false sinon.

♦ Partie I : Une approche récursive

Dans cette partie, on utilise une approche récursive.

1. Ecrire une fonction supprime_premier qui prend en argument un caractère car et une chaine et renvoie la chaine obtenue en supprimant la premier occurence de car dans chaine.

Par exemple supprime_premier("c","niche") renvoie "nihe". Si car n'est pas dans chaine alors on renvoie chaine sans modification.

Par exemple supprime_premier("1","Python") renvoie "Python"

```
def supprime_premier(car, chaine):
       res = ''
2
        # On parcourt la chaine en la recopiant et
3
        # on s'arrête la première fois qu'on rencontre le caractère à supprimer
       for i in range(len(chaine)):
            if chaine[i]!=car:
6
                res = res + chaine[i]
            else:
                break
        #On copie le reste de la chaine
10
       res = res + chaine[i+1:]
1.1
       return res
12
```

2. Ecrire une fonction récursive anagrammes_rec qui prend en argument deux chaines de caractères et renvoie True si ce sont des anagrammes l'une de l'autre et False sinon.

Par exemple, anagrammes_rec("niche", "chien") renvoie True.

```
def anagrammes_rec(chaine1,chaine2):
    if len(chaine1)==0 and len(chaine2)==0:
        return True
    nchaine2 = supprime_premier(chaine1[0],chaine2)
    # Si la suppression n'a rien donné, alors ce ne sont pas des anagrammes
    if len(nchaine2)==len(chaine2):
        return False
    return anagrammes_rec(chaine1[1:],nchaine2)
```

3. Donner (en la justifiant) la complexité de cette fonction en notant n la taille de la plus longue des deux chaines.

On effectue un appel récursif pour chaque caractères de la chaine1 donc au plus n appels. Pour chacun de ses appels, on parcourt la chaine2 (donc au moins n opérations). On total, la complexité est donc $O(n^2)$.

♦ Partie II : Une approche itérative

Dans cette partie, on utilise une approche itérative en manipulant les dictionnaires de Python.

1. Ecrire une fonction cree_dico qui prend en argument une chaine de caractères et renvoie un dictionnaire dont les clés sont les caractères composant la chaine et les valeurs leur nombre d'apparition.

Par exemple, cree_dico("epele") renvoie le dictionnaire {'e':3, 'p':1, 'l':1} en effet dans le mot 'epele', 'e' apparaît à trois reprises et 'l' et 'p' chacun une fois.

```
def cree_dico(chaine):
    dico = {}

for c in chaine:
    if c in dico:
        #caractere déjà présent, on ajoute 1 à son nombre d'occurence
    dico[c] = dico[c]+1

else:
    #caractere qui apparait pour la première fois
    dico[c] = 1

return dico
```

2. Ecrire une fontionc egaux qui prend en argument deux dictionnaires et renvoie True si ces deux dictionnaires sont égaux (c'est à dire contiennent exactement les mêmes clés avec les mêmes valeurs) et False sinon

Par exemple, egaux({'e':3, 'p':1, 'l':1 },{'p':1,'e':2,'l':2}) renvoie False

A on s'interdit ici d'utiliser le test d'égalité == entre deux dictionnaires et on écrira un parcours de dictionnaire.

```
def egaux(dico1,dico2):
    if len(dico1)!=len(dico2):
        return False
    for cle in dico1:
        if (cle not in dico2 or dico1[cle]!=dico2[cle]):
            return False
    return True
```

3. Ecrire une fonction anagrammes_iter qui prend en argument deux chaines de caractères et renvoie True si ce sont des anagrammes et False sinon.

```
def anagrammes_iter(chaine1,chaine2):
    dico1 = cree_dico(chaine1)
    dico2 = cree_dico(chaine2)
    return egaux(dico1,dico2)
```

4. Donner (en la justifiant) la complexité de cette fonction en notant n la taille de la plus longue des deux chaines.

On sait que le test d'appartenance à un dictionnaire est en O(1), donc ici la création de chacun des deux dictionnaires est en O(n) (car on n'a que des opérations en O(1) dans la boucle). De même le test d'égalité entre les deux dictionnaires est en O(n) et donc cette fonction est en O(n).

□ Exercice 3 : gestion de tests dans une entreprise

d'après ccinp 2023 - TPC, TSI

❖ Partie I : Tests de code de sécurité sociale

En France, le numéro de sécurité sociale se compose de 15 chiffres, comme détaillé sur l'image ci-dessous, les deux derniers chiffres sont calculés à partir des 13 premiers et forment une clé de contrôle destinée à vérifier que le numéro est valide. Le but de l'exercice est d'écrire un programme Python effectuant cette vérification.



Credits: service-public.fr

Pour calculer la valeur de la clé de contrôle :

- on calcule le reste dans la division euclidienne du nombre formé par les 13 premiers chiffres par 97,
- On soustrait de 97 le résultat obtenu.

Par exemple pour le numéro figurant sur l'image ci-dessus :

- on calcule le reste dans la division euclidienne de 2 94 03 75 120 005 (les 13 premiers chiffres), on trouve 6.
- On soustrait de 97 le résultat obtenu : 97 6 = 91.

La clé de contrôle est dans ce cas 91, et donc le numéro ci-dessus est non valide puisque sa clé de contrôle est 22.

On suppose que les numéros de sécurité sociale sont fournis sous la forme de chaines de caractères (type str de Python) contenant des chiffres et des espaces, par exemple le numéro complet ci-dessous est donné sous la forme de la chaine "2 94 03 75 120 005 22" et le numéro sans la clé de contrôle par "2 94 03 75 120 005". Et on rappelle qu'en Python, on peut toujours convertir une chaine de caractères en entier (type int) et inversement.

1. Ecrire la fonction num_secu qui prend en argument une chaine de caractères représentant un numéro de sécurité sociale (avec ou sans la clé), supprime les espaces et renvoie le nombre entier formé par les chiffres de ce numéro de sécurité sociale.

Par exemple num_secu("2 94 03 75 120 005") renvoie l'entier 2940375120005

```
def num_secu(chaine):
    ns = ""

# suppresion des espaces
for c in chaine:
    if c!=" ":
        ns = ns + c
return int(ns)
```

2. Ecrire la fonction clef qui détermine la clé de contrôle d'un numéro de sécurité sociale à 13 chiffres. Cette fonction prend donc un paramètre de type int et renvoie un int.

Par exemple clef (2940375120005) renvoie l'entier 91

```
def clef(ns_debut):
return 97-(ns_debut%97)
```

3. Ecrire la fonction num_secu_complet qui détermine le numéro de sécurité sociale complet à partir des 13 premiers chiffres. Cette fonction doit donc calculer la clé de contrôle et l'ajouter à la fin du numéro de sécurité social donné en argument.

Par exemple num_secu_complet(2940375120005) renvoie l'entier 294037512000591

```
def num_secu_complet(ns_debut):
    return ns_debut*100 + clef(ns_debut)
```

4. Ecrire la fonction test_num_secu qui détermine si le numéro de sécurité social complet donné en argument (sous la forme d'une chaine de caractères) est correct.

Par exemple, test_num_secu("2 94 03 75 120 005 22") renvoie False.

```
def test_num_secu(ns):
    ns_complet = num_secu(ns)
    ns_debut = num_secu(ns[:-2])
    return num_secu_complet(ns_debut) == ns_complet
```

❖ Partie II : Tests de numéro de carte de crédit

Comme pour les numéros de sécurité de sociale, un algorithme (appelé algorithme de Luhn), permet de vérifier qu'un numéro de carte de crédit est valide. Les étapes sont les suivantes :

- on commence par extraire du numéro la liste des chiffres de rang impair ainsi que celle des chiffres de rang pair, en numérotant les chiffre à partir de la droite. Par exemple, sur le numéro 437716 cette procédure donne [3, 7, 6] pour les chiffres de rang impair et [1, 7, 4] pour ceux de rang pair.
- On double ensuite chaque chiffre de la liste des rangs pairs et si on obtient un chiffre plus grand que 9, alors on le remplace par la somme des deux chiffres qui le compose. Dans l'exemple précédent, la liste des chiffres de rang pair [1, 7, 4] devient donc [2, 5, 8] car 14 est remplacé par la somme de ses chiffres donc 5.
- On calcule ensuite la somme des chiffres des deux listes, si le résultat obtenu est divisible par 10 alors le numéro de la carte de crédit est valide. Dans l'exemple précédent, on calcule donc : :

```
3 + 7 + 6 + 2 + 5 + 8 = 33,
```

et comme 33 n'est pas divisible par 33, le numéro n'est pas valide.

1. Vérifier que le numéro 4762 est valide.

La liste des chiffres de rang impair est [2, 7] et celle de rang pair est [6, 4]. Après traitement, celle de rang pair devient [3, 8]. On additionne l'ensemble des chiffres obtenus : 2+7+3+8=20, on obtient bien un entier divisible par 10, donc le numéro est valide.

2. Ecrire une fonction num_en_liste qui prend en argument un entier et renvoie la liste de ses chiffres. Par exemple, num_en_liste(4762) renvoie la liste [4, 7, 6, 2].

```
def num_en_liste(n):
    lchiffres = []
    for c in str(n):
        lchiffres.append(int(c))
    return lchiffres
```

3. Ecrire une fonction impairs_pairs qui prend en argument une liste 1 et renvoie deux listes, celles des éléments d'indice impair de 1 et celle éléments d'indice pairs en numérotant les indices à partir de la droite. Par exemple pairs_impairs([4, 7, 6, 2]) renvoie [2, 7] et [6, 4]

```
def impairs_pairs(lchiffres):
    limpairs = [lchiffres[i] for i in range(len(lchiffres)-1,-1,-2)]
    lpairs = [lchiffres[i] for i in range(len(lchiffres)-2,-1,-2)]
    return limpairs, lpairs
```

4. Ecrire une fonction traite_pairs qui prend en argument une liste 1, ne renvoie rien et modifie cette liste en remplaçant chaque chiffre de la liste par son double. Si le résultat obtenu est supérieur à 9 alors il faut le remplacer par la somme des deux chiffres qui le composent. Par exemple, si 1=[6, 4], après l'appel traite_pairs[1], le contenu de 1 devient [3, 8] en effet 4 a été remplacé par son double 8 et 6 par la somme des chiffres de son double 12.

```
def traite_pairs(pairs):
    res = []
for p in pairs:
    if p*2 < 10:
        res.append(p*2)
    else:
        res.append((2*p)//10 + (2*p)%10)
    return res</pre>
```

5. Ecrire une fonction test_num_carte qui prend en argument un entier et renvoie True si c'est le numéro d'une carte de crédit valide et False sinon. Par exemple, test_num_carte(4762) renvoie True.

```
def test_num_cart(num):
    lchiffres = num_en_liste(num)
    limpairs, lpairs = impairs_pairs(lchiffres)
    lpairs = traite_pairs(lpairs)
    s = 0
    for c in limpairs:
        s = s + c
    for c in lpairs:
        s = s + c
    return s%10 == 0
```

☐ Exercice 4 : requête SQL sur une seule table

On considère la base de données pays_du_monde contenant une seule table pays dont le schéma est donné ci-dessous :

pays		
nom	:	TEXT
region	:	TEXT
population	:	INT
surface	:	INT
cotes	:	INT
pib	:	INT

D'autre part, on précise la signification des champs suivants :

- population: le nombre d'habitants du pays.
- region : La région du pays (par exemple "europe de l'ouest")
- area : la surface du pays (en km carré).
- coastline : la surface côtière du pays, cette valeur vaut 0 lorsque le pays n'a pas d'ouverture sur la mer
- pib : le produit intérieur brut par habitant, c'est une mesure de la richesse du pays.

Et on indique que la requête SELECT DISTINCT region FROM pays a renvoyé le résultat suivant :

region		
Asie		
Afrique du nord		
Europe de l'est		
Europe de l'ouest		
Oceanie		
Afrique sub saharienne		
Proche orient		
Amérique latine		
Amérique du nord		

Ecrire les requêtes permettant de :

1. Trouver la population et le produit intérieur brut de la France.

```
SELECT population, pib
FROM pays
WHERE nom = "France";
```

2. Trouver les pays d'europe n'ayant pas d'ouverture sur la mer.

```
SELECT nom
FROM pays
WHERE cotes=0 and (region="Europe de l'est" or region = "Europe de l'ouest");
```

3. Classer par ordre alphabétique les pays d'amérique latine.

```
SELECT nom
FROM pays
WHERE region="Amérique latine"
ORDER BY nom;
```

4. Lister par ordre décroissant du nombre d'habitants les cinq pays les plus peuplés

```
SELECT nom
FROM pays
ORDER BY population DESC
LIMIT 5;
```

5. Trouver le pays du proche orient le plus riche (ayant le pib le plus élevé).

```
SELECT nom, MAX(pib)
FROM pays
WHERE region="Proche orient";
```

6. Classer le pays d'afrique du nord par densité décroissante de population (la densité est le rapport entre le nombre d'habitant et la surface)

```
SELECT nom, population/surface AS densite
FROM pays
WHERE region="Afrique du nord"
ORDER BY densite DESC;
```

7. Classer les régions par somme du pib décroissante des pays qui les composent.

```
SELECT region, SUM(pib) as total_pib
FROM pays
GROUP BY region
ORDER BY total_pib DESC;
```