Concours National Commun (CNC), Maroc

Session 2022, filière TSI

Informatique

Durée : 2 heures

Ce que traite le sujet

- Exercice : approximation du nombre pi
- Partie I : base de données et SQL
- Partie II : calcul numérique
- Partie III : problème

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/cnc/tsi/2022/info

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

ⵜⴰⴳⵍⴷⵉⵜ ⵏ ⵍⵎⵖⵔⵉⴱ
ⵜⴰⵎⴰⵡⴰⵙⵜ ⵏ ⵜⵣⵔⴼⵜ ⴰⵏⴰⵎⵓⵔ
ⴰⵙⴳⵎⵉ ⴰⵙⵙⵏⵜⵉⵙ ⴰⵏⴰⵎⵓⵔ

المملكة المغربية
وزارة التعليم العالي
والبحث العلمي والابتكار

ROYAUME DU MAROC
Ministère de l'Enseignement Supérieur, de la Recherche Scientifique et de l'Innovation

المدرسة الوطنية العليا للمعادن بالرباط
ⵜⵜⴰⵙⵙⵓⵔⵜ ⵜⵓⵏⴰⵢⵜ ⵜⵓⵏⵉⵙⵜ ⵏ ⵍⵎⵄⴰⴷⵏ ⵖ ⵕⵕⴱⴰⵟ
ECOLE NATIONALE SUPERIEURE DES MINES DE RABAT

وزارة الانتقال الطاقي
Ministère de la Transition Énergétique
والتنمية المستدامة
et du Développement Durable
Département de la Transition Energétique
قطاع الانتقال الطاقي

MINES-RABAT

CNC 2022

Concours National Commun

d'Admission dans les Établissements de Formation d'Ingénieurs et Établissements Assimilés

Épreuve d'**INFORMATIQUE**

Filière : **TSI**

Durée **2 heures**

Cette épreuve comporte **10 pages au format A4**, en plus de la page de garde

Les candidats sont informés que la précision des raisonnements algorithmiques ainsi que le soin apporté à la rédaction et à la présentation des copies seront des éléments pris en compte dans la notation. Il convient en particulier de rappeler avec précision les références des questions abordées. Si, au cours de l'épreuve, un candidat repère ce qui peut lui sembler être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre

Remarques générales :

- ✓ Cette épreuve est composée d'un exercice et de trois parties tous indépendants ;
- ✓ Toutes les instructions et les fonctions demandées seront écrites en Python ;
- ✓ Les questions non traitées peuvent être admises pour aborder les questions ultérieures ;
- ✓ Toute fonction peut être décomposée, si nécessaire, en plusieurs fonctions.

≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈

Important : ***Le candidat doit impérativement commencer par traiter toutes les questions de l'exercice ci-dessous, et écrire les réponses dans les premières pages du cahier de réponses.***

Exercice : (4 points)

Approximation du nombre π

La constante **PI** (**π**) est l'un des nombres les plus importants et les plus fascinants dans le monde des mathématiques. C'est un nombre irrationnel, son écriture est faite d'une infinité de décimales, sans aucune périodicité. On n'arrive qu'à déterminer une écriture décimale approchée de sa valeur réelle. Pour cela, plusieurs formules pratiques et efficaces ont été employées.

Le mathématicien suisse *Leonard EULER* est à l'origine d'une formule qui donne des valeurs approchées du nombre **π**, de plus en plus précises. La formule d'Euler est la suivante :

$$\pi \approx 2 * \sum_{k=0}^{n} \frac{k!}{1 * 3 * 5 * \ldots * (2k+1)}$$

Q1- Écrire la fonction ***fact (k)*** qui reçoit en paramètre un entier positif **k**, et qui retourne la valeur de factorielle **k** : **k! = 1*2*3* … *k.**

Q2- Écrire la fonction ***produit (k)*** qui reçoit en paramètre un entier positif **k**, et qui retourne la valeur du produit suivant :

$$1 * 3 * 5 * \ldots * (2k+1)$$

Q3- Écrire la fonction ***terme (k)*** qui reçoit en paramètre un entier positif **k**, et qui retourne la valeur de l'expression suivante :

$$\frac{k!}{1 * 3 * 5 * \ldots * (2k+1)}$$

Q4- Écrire la fonction ***list_termes (n)*** qui reçoit en paramètre un entier positif **n**. La fonction retourne une liste ***L*** qui contient les valeurs de **terme (k)** pour **k= 0, 1, 2, 3, …, n.**

Q5- Écrire la fonction ***somme (L)*** qui reçoit en paramètre une liste **L** de nombres réels, et qui retourne la valeur de la somme des éléments de **L**, sans utiliser la fonction prédéfinie sum().

Q6- Écrire la fonction ***pi_euler (n)*** qui reçoit en paramètre un entier positif **n**, et qui retourne la valeur approchée de **π**, en utilisant la formule d'Euler.

Partie I : Base de données et langage SQL

Internationaux de tennis

On considère une base de données qui permet la gestion des joueurs de tennis, ayant gagné au moins un tournoi international de tennis simple. Cette base de données est composée de 3 tables : **'Joueur'**, **'Gagne'** et **'Tournoi'**.

Joueur	
id	*(entier)*
nomJr	*(texte)*
genre	*(texte)*
nationalité	*(texte)*

Gagne	
idJr	*(entier)*
codTr	*(entier)*
année	*(entier)*
prime	*(réel)*

Tournoi	
code	*(entier)*
nomTr	*(texte)*
catégorie	*(texte)*
pays	*(texte)*

a- Structure de la table 'Joueur'

La table **'Joueur'** contient des informations sur les joueurs ayant gagné au moins un tournois international de tennis simple : Le champ **id** est la clé primaire, ce champ contient un entier unique pour identifier chaque joueur. Le champ **nomJr** contient le nom complet de chaque joueur. Le champ **genre** contient **'F'** pour féminin, ou **'M'** pour masculin. Le champ **nationalité** contient le pays d'origine de chaque joueur.

Exemples :

id	nomJr	genre	nationalité
2154	Roger Federer	M	Suisse
3432	Serena Williams	F	États Unis
2691	Novak Djokovic	M	Serbie
2374	Rafael Nadal	M	Espagne
...	...	...	...

b- Structure de la table 'Tournoi'

La table **'Tournoi'** contient des informations sur les tournois internationaux de tennis simples. Le champ **code** est la clé primaire, ce champ contient un entier unique pour chaque tournoi. Les champs **nomTr** et **catégorie** contiennent le nom et la catégorie de chaque tournoi. Le champ **pays** contient le pays dans lequel se déroule chaque tournoi.

Exemples :

code	nomTr	catégorie	pays
23	Roland-Garros	Grand Chelem	France
174	Doha	ATP 250	Qatar
10	Open d'Australie	Grand Chelem	Australie
59	Sydney International	ATP 250	Australie
85	Rotterdam	ATP 500	Pays-Bas
96	Masters d'Indiana Wells	Masters 1000	États Unis
...	...	...	...

c- Structure de la table 'Gagne'

La table **'Gagne'** contient des informations sur les joueurs et les tournois gagnés par chaque joueur. Les champs **idJr** et **codTr** sont deux clés étrangères qui font respectivement référence aux clés primaires **Id** et **code**. Le champ **année** contient l'année de chaque tournoi. Chaque tournoi est organisé une seule fois chaque année. Le champ **prime** contient le montant reçu par le joueur gagnant de chaque tournoi, et il est exprimé en million de dollars. La prime d'un tournoi peut changer au cours des années.

Exemples :

idJr	codTr	année	prime
2374	23	2019	2.30
2374	23	2020	1.60
2374	23	2021	1.40
3432	85	2018	2.50
2691	10	2020	4.12
2691	10	2021	2.75
...	...	...	...

- **Q.1** – Rédiger une requête SQL qui ajoute, dans la table **'Tournoi'**, un nouveau tournoi dans la catégorie **'ATP World Tour'**, et qui se déroule dans le pays **'Émirats Arabes Unis'**. Le code du tournoi est **5741**, son nom est **'Dubai'**.

- **Q.2** – Rédiger une requête SQL qui permet d'obtenir les différentes catégories des tournois, triées dans l'ordre alphabétique.

- **Q.3** – Rédiger en algèbre relationnelle, une requête qui permet d'obtenir les noms des tournois qui se sont déroulés au pays **'Australie'** durant l'année **2021**, et les noms des joueurs qui ont gagné ces tournois.

- **Q.4** – Écrire la requête **Q.3** en langage SQL.

- **Q.5** – Rédiger une requête SQL, qui permet d'obtenir les codes des tournois, la plus grande prime, la plus petite prime et la moyenne des primes de chaque tournoi, triés dans l'ordre décroissant de la moyenne des primes.

Q.6 – Rédiger une requête SQL, qui permet d'obtenir les noms et genres des joueurs, le compte des tournois gagnés par chaque joueur et la somme totale des primes reçue par chaque joueur. La somme totale des primes reçue par chaque joueur doit être supérieure à **75** millions de dollars.

Q.7 – Rédiger une requête SQL qui permet d'obtenir les années et les noms et les genres des joueurs qui ont gagné les deux tournois **'Roland-Gaross'** et **'Wembledon'** la même année, triés dans l'ordre décroissant des années.

Partie II : Calcul numérique

Recherche du zéro d'une fonction

Méthode 'Dekker'

Dans cette partie, on suppose que les modules **numpy** et **matplotlib.pyplot** sont importés :

```
import numpy as np
import matplotlib.pyplot as pl
```

Calculer les zéros d'une fonction réelle f (c'est-à-dire les racines de l'équation $(x) = 0$) est un problème que l'on rencontre très souvent en calcul scientifique. En général, cette tâche ne peut être effectuée en un nombre fini d'opérations. Par exemple, il n'existait pas de formule explicite donnant les racines d'un polynôme quelconque de très grand degré. La situation est bien sûr encore plus complexe quand f n'est pas un polynôme.

Pour résoudre ce problème, on utilise donc des méthodes itératives ou récursives : partant d'une ou plusieurs valeurs initiales, on construit une suite de valeurs x_n qui converge vers un zéro de la fonction f considérée.

Un problème simple et concret qui donne lieu à une équation non linéaire est celui de l'**équation d'état d'un gaz** : Nous voulons déterminer le volume V occupé par un gaz dont la température est T et dont la pression est p. L'équation d'état, (liant p, V et T) est donnée par :

$$\left(p + a\frac{N^2}{V^2}\right) * (V - N.b) = k.N.T$$

Avec :

- a et b sont deux coefficients qui dépendent du gaz considéré ;
- N est le nombre de molécules contenues dans le volume V ;
- p est la pression ;
- T est la température ;
- k est la constante de Boltzmann.

Nous devons donc résoudre une équation non linéaire dont la solution est V. Pour cela, on doit calculer les zéros de la fonction g définit par :

$$g(v) = \left(p + \frac{a.N^2}{v^2}\right) * (v - N.b) - k.N.T$$

On suppose que les valeurs des paramètres sont les suivantes, (*pour le dioxyde de carbone CO_2*) :

a=0.40 , b=42.7 * 10^{-6} , N=10^3 , T=300 , p=3.5*10^7 et k=1.38*10^{-23}

Q.1- Écrire la fonction $g(v)$ en langage Python.

Q.2- Écrire le programme python qui permet de tracer la représentation graphique ci-dessous (Figure 1), de la fonction g. Le nombre de points générés dans la courbe est **500**.

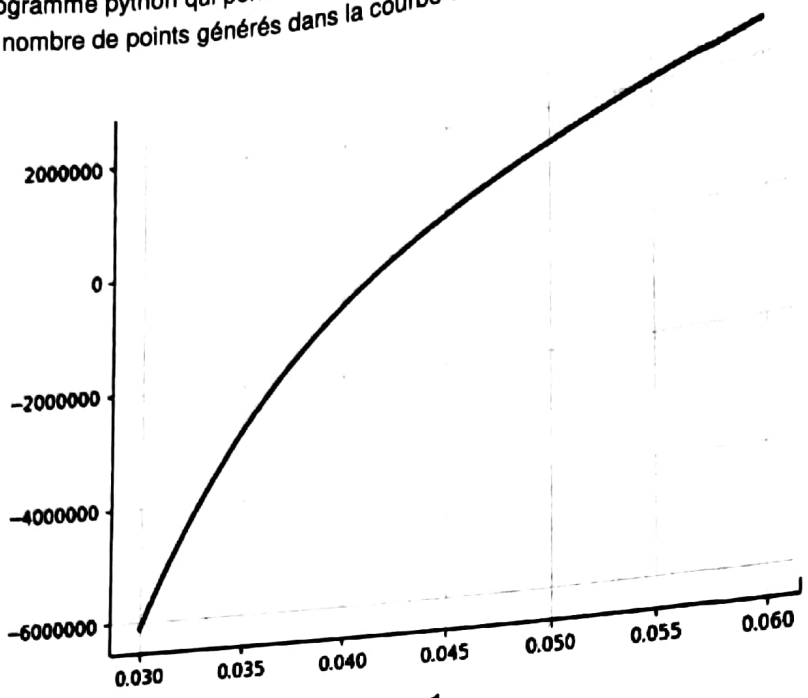


Figure 1

À partir de cette représentation graphique, on voit bien que la fonction g admet un zéro dans l'intervalle : **[0.040 , 0.045]**.

La méthode de dichotomie :

La méthode de la dichotomie (ou de la bissection) est un algorithme de recherche d'un zéro d'une fonction. Cette méthode consiste à répéter de partager un intervalle en deux parties, et sélectionner le sous-intervalle dans lequel existe le zéro de la fonction.

La méthode de la sécante :

La méthode de la sécante est un algorithme de recherche d'un zéro d'une fonction. Cette méthode est basée sur la relation de récurrence suivante :

Étant donnés deux réels différents a et b proches du zéro d'une fonction f :

$$c = b - f(b)\frac{b-a}{f(b)-f(a)}$$

Le nombre c est la valeur de l'abscisse du point d'intersection entre l'axe des abscisses et la droite passante par les deux points de coordonnées $(a, f(a))$ et $(b, f(b))$.

La méthode de 'Dekker' :

La méthode de **Dekker** est un algorithme de recherche d'un zéro d'une fonction, combinant les deux méthodes précédentes : la dichotomie et la sécante. L'idée d'allier ces deux méthodes différentes est due à *Theodorus Dekker (1969)*.

La méthode de Dekker est basée sur le principe suivant : À chaque itération, elle décide laquelle des deux méthodes (la dichotomie et la sécante) est susceptible d'approcher au mieux le zéro de la fonction, et effectue une itération en utilisant cette méthode. *L'idée principale est d'utiliser la méthode de la sécante parce qu'elle converge vite, et de revenir à la méthode de dichotomie si besoin est.*

Algorithme de la méthode de 'Dekker' :

Entrée : f est une fonction continue ayant un zéro dans un intervalle $[a\,;\,b]$, telle que $f(a)$ *et* $f(b)$ ont des signes opposés, et **eps** est la précision désirée.

Tant que $f(b) \neq 0$ *et* $|b - a| > eps$ *(condition de convergence)*

- $m \leftarrow$ la valeur du milieu de a et b
- $c \leftarrow$ la valeur de la formule de la sécante passante par les deux points $(a, f(a))$ et $(b, f(b))$
- Si d *est compris* entre b et m, alors
  - $d \leftarrow c$
- Sinon
  - $d \leftarrow m$
- Fin si
- Si $f(b)$ *et* $f(d)$ ne sont pas de même signe, alors
  - $a \leftarrow b$.
- Fin si
- $b \leftarrow d$
- Si $|f(a)| < |f(b)|$ alors
  - échanger les valeurs de a et b
- Fin si

Fin tant que

Sortie : retourner b

Q.3- Écrire la fonction ***secante*** (f, a, b), qui reçoit en paramètres une fonction f et deux réels a et b tels que $f(a)$ *et* $f(b)$ ont des signes opposés. La fonction retourne la valeur c de l'abscisse du point d'intersection entre l'axe des abscisses et la droite passante par les deux points de coordonnées $(a, f(a))$ *et* $(b, f(b))$.

Q.4- Écrire la fonction ***dekker_racine*** (f, a, b, eps), qui reçoit en paramètres une fonction f continue ayant un zéro dans un intervalle $[a\,;\,b]$ tellee que $f(a)$ *et* $f(b)$ ont des signes opposés, et un réel **eps** qui représente la précision. En utilisant l'algorithme de la méthode de Dekker (ci-dessus), la fonction retourne la valeur du zéro de la fonction f.

Q.5- Écrire le programme python qui permet d'afficher la valeur du zéro de la fonction g (voir Figure1), en utilisant la méthode de Dekker, sachant que la valeur de la précision choisie est : **eps** = 10^{-12}

Partie III : Problème

Résolution de Sudoku

Méthode 'Back-tracking'

Qu'est-ce qu'un Sudoku ?

Le **Sudoku** (**Su** : Nombre , **Doku** : Unique) es un jeu de nombre très populaire. Ce jeu est constitué d'une matrice carrée de **9** lignes et **9** colonnes, divisée en **9** sections de **3*3** cases chacune.

Le principe du jeu est simple : remplir les cases de la matrice avec des nombres compris entre **1** et **9**. Attention il est obligatoire de respecter les **3** règles suivantes : un nombre doit apparaitre une seule fois dans chaque ligne, dans chaque colonne et dans chaque section.

Exemple :

Section 3*3

Section 3*3

5	3	9	8	7	6	4	1	2
7	2	8	3	1	4	9	6	5
6	4	1	2	9	5	7	3	8
4	6	2	5	3	9	8	7	1
3	8	5	7	2	1	6	4	9
1	9	7	4	6	8	2	5	3
2	5	6	1	8	7	3	9	4
9	1	3	6	4	2	5	8	7
8	7	4	9	5	3	1	2	6

Pour représenter la matrice du sudoku, on utilise une liste composée de **9** listes toutes de même taille **9**.

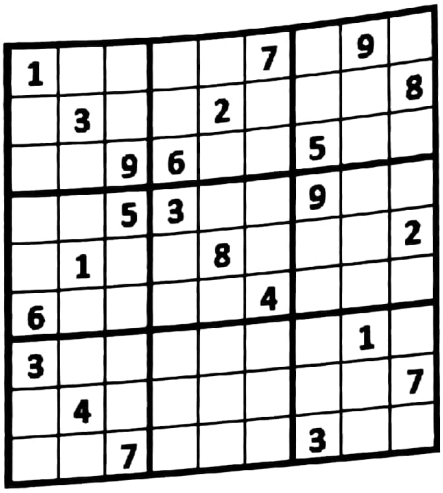
Par exemple, la matrice sudoku de l'exemple ci-dessus est représentée par la liste **S** suivante :

```
S = [ [ 5, 3, 9, 8, 7, 6, 4, 1, 2 ],
      [ 7, 2, 8, 3, 1, 4, 9, 6, 5 ],
      [ 6, 4, 1, 2, 9, 5, 7, 3, 8 ],
      [ 4, 2, 6, 5, 3, 9, 8, 7, 1 ],
      [ 3, 8, 5, 7, 2, 1, 6, 4, 9 ],
      [ 1, 9, 7, 4, 6, 8, 2, 5, 3 ],
      [ 2, 5, 6, 1, 8, 7, 3, 9, 4 ],
      [ 9, 1, 3, 6, 4, 2, 5, 8, 7 ],
      [ 8, 7, 4, 9, 5, 3, 1, 2, 6 ] ]
```

Jeu de sudoku

Au début du jeu, la matrice de sudoku aura des cases déjà remplies. La tâche du joueur consistera à se baser sur la logique pour remplir les nombres manquants et compléter progressivement.la matrice, tout en respectant les **3** règles citées avant.

Exemple :



Dans la matrice sudoku initiale, on utilisera le nombre **0** pour représenter chaque case vide. Ainsi la matrice sudoku initiale de l'exemple ci-dessus est représentée par la liste **S** suivante :

```
S = [ [ 1,0,0,0,0,7,0,9,0 ],
      [ 0,3,0,0,2,0,0,0,8 ],
      [ 0,0,9,6,0,0,5,0,0 ],
      [ 0,0,5,3,0,0,9,0,0 ],
      [ 0,1,0,0,8,0,0,0,2 ],
      [ 6,0,0,0,0,4,0,0,0 ],
      [ 3,0,0,0,0,0,0,1,0 ],
      [ 0,4,0,0,0,0,0,0,7 ],
      [ 0,0,7,0,0,0,3,0,0 ] ]
```

NB : On utilisera cette matrice ***S*** dans tous les exemples des fonctions de cette partie.

Une matrice sudoku initiale incomplète correctement constituée, ne peut aboutir qu'à une et une seule solution. Le nombre de matrices sudku (à solution unique) complètes possibles est de $6.67*10^{21}$. Et si l'on considère que deux matrices sont identiques, lorsque l'on peut arriver de l'une à l'autre par des opérations matricielle, alors le nombre de matrices différentes est de **5 472 730**.

Il existe de nombreuses méthodes de résolution du jeu de sudoku. Parmi ces méthodes, on trouve la méthode du '**back-tracking**'. Cette méthode consiste à remplir les cases vides : case par case. Et si on découvre un conflit avec les **3** règles du jeu, on est obligé de revenir en arrière. Le retour en arrière est considérablement simplifié avec l'usage de la **récursivité**.

Dans la suite de cette partie, on considère que ***S*** est une variable globale déclarée et initialisée par une matrice sudoku initiale incomplète, ayant une solution unique.

Q.1- Écrire la fonction ***ligne (l)*** qui reçoit en paramètre un entier positif ***l*** qui représente l'indice d'une ligne de la matrice sudoku initiale incomplète ***S***. La fonction retourne la liste des nombres qui apparaissent dans les cases non vides de la ligne d'indice ***l*** dans ***S***.

Exemples :

- ✓ La fonction **ligne (0)** retourne la liste **[1, 7, 9]**
- ✓ La fonction **ligne (5)** retourne la liste **[6, 4]**

Q.2- Écrire la fonction ***colonne (j)*** qui reçoit en paramètre un entier positif ***j*** qui représente l'indice d'une colonne de la matrice sudoku initiale incomplète ***S***. La fonction retourne la liste des nombres qui apparaissent dans les cases non vides de la colonne d'indice ***j*** dans ***S***.

Exemples :

- ✓ La fonction **colonne (1)** retourne la liste **[3, 1, 4]**
- ✓ La fonction **colonne (7)** retourne la liste **[9, 1]**

Q.3- Écrire la fonction ***section (i, j)*** qui reçoit en paramètres deux entier positifs ***i*** et ***j*** qui représentent respectivement les indices de la ligne et de la colonne d'une case vide dans la matrice sudoku initiale incomplète ***S***. La fonction retourne la liste des nombres qui apparaissent dans les cases non vides de la section à laquelle appartient la case ***(i, j)***.

Exemple :

1					7		9	
	3			2				8
		9	6			5		
		5	3			9		
	1			8				2
6					4			
3							1	
	4							7
		7				3		

- ✓ La fonction **section (1, 0)** retourne la liste **[1, 3, 9]**
- ✓ La fonction **section (5, 7)** retourne la liste **[9, 2]**

Q.4- Écrire la fonction ***cases_vides ()*** qui retourne la liste ***L*** contenant toutes les cases vides dans la matrice sudoku initiale incomplète ***S***. Chaque case est représentée par un tuple qui contient l'indice de la ligne et l'indice de la colonne de cette case.

Exemple :

La fonction **cases_vides ()** retourne la liste ***L*** suivante :

L = [(0, 1), (0, 2), (0, 3), (0, 4), (0, 6), (0, 8), (1, 0), (1, 2), (1, 3), (1, 5), (1, 6), (1, 7), (2, 0), (2, 1), (2, 4), (2, 5), (2, 7), (2, 8), (3, 0), (3, 1), (3, 4), (3, 5), (3, 7), (3, 8), (4, 0), (4, 2), (4, 3), (4, 5), (4, 6), ... , (8, 7), (8, 8)]

Étant donnée une case vide ***(i, j)*** dans la matrice ***S***. Un nombre est **candidat** pour cette case si ce nombre n'existe pas déjà dans toutes les cases de la ligne d'indice ***i***, ni dans toutes les cases de la colonne d'indice ***j***, ni dans toutes les cases de la section à laquelle appartient la case ***(i, j)***.

Q.5 - Écrire la fonction ***candidats (t)*** qui reçoit en paramètre un tuple ***t*** qui contient l'indice de la ligne et l'indice de la colonne d'une case vide dans la matrice sudoku initiale incomplète ***S***. La fonction retourne la liste **C** contenant les nombres candidats pour la case vide ***t***.

Exemples :

- ✓ La fonction **candidats ((0, 1))** retourne la liste **C= [2, 5, 6, 8]** *(la case (0,1) possède 4 candidats)*
- ✓ La fonction **candidats ((4, 7))** retourne la liste **C= [3, 4, 5, 6, 7]** *(la case (4,7) possède 5 candidats)*

Afin de réduire la complexité de la méthode du 'back-tracking', il faut réduire au maximum le nombre de possibilités : plus le nombre de possibilités est important plus les risques d'erreur et retour en arrière tardif sont nombreux. Pour minimiser le risque d'erreur et donc le nombre d'opérations réalisées, il faut déterminer un ordre de parcours des cases vides dans la matrice sudoku, en remplissant les cases ayant le moins de possibilités, aux cases ayant le plus de possibilités.

Pour cela, on a besoin de trier les cases vides, dans l'ordre croissant des comptes de candidats à chaque case vide.

Q.6- Écrire la fonction ***tri_cases (L)*** qui reçoit en paramètre la liste ***L*** des cases vides dans la matrice sudoku initiale ***S***. La fonction trie les cases de ***L*** dans l'ordre croissant des comptes de candidats à chaque case.

Exemple :

Après l'appel de la fonction **tri_cases (L)**, on obtient la liste suivante :

[(1, 2), (4, 2), (0, 3), (0, 4), (0, 6), (0, 8), (1, 0), (1, 5), (1, 7), (2, 1), (2, 4), (2, 5), (2, 8), (3, 1), (3, 4), (3, 5), (3, 8), (4, 0), (4, 3), (4, 5), (4, 6), (5, 2), (5, 6), (5, 8), (6, 2), (7, 6), (0, 1), (0, 2), (1, 3), (1 , 6), ... , (8, 5), (7, 5)]

La liste ***L*** contient les cases vides dans la matrice sudoku initiale incomplète. Les cases de ***L*** sont triées dans l'ordre croissant des nombres de candidats pour chaque case. Pour trouver le nombre correspondant à chaque case vide, on utilise l'algorithme récursif suivant, basé sur la méthode du Back-tracking :

Pour chaque case vide ***t*** de la liste ***L*** :

Parcourir les éléments **x** de la liste **C** des nombres candidats à la case ***t*** :

- Placer l'élément **x** dans la case ***t***, et passer récursivement à la case suivante dans **L** ;
- S'il n'y a pas de conflit avec les **3** règles de sudoku, la fonction retourne **True**
- Sinon, placer dans la case ***t*** l'élément suivant à **x** dans la liste **C**.

Q.7- Écrire la fonction ***solution (L)*** qui place le nombre correspondant à chaque case vide dans la matrice sudoku initiale incomplète ***S***.

Exemple :

L'appel de la fonction **solution (L)** permet de compléter la matrice sudoku initiale :

```
S = [ [ 1,6,2,8,5,7,4,9,3 ],
      [ 5,3,4,1,2,9,6,7,8 ],
      [ 7,8,9,6,4,3,5,2,1 ],
      [ 4,7,5,3,1,2,9,8,6 ],
      [ 9,1,3,5,8,6,7,4,2 ],
      [ 6,2,8,7,9,4,1,3,5 ],
      [ 3,5,6,4,7,8,2,1,9 ],
      [ 2,4,1,9,3,5,8,6,7 ],
      [ 8,9,7,2,6,1,3,5,4 ] ]
```

********* **FIN** *********

Travailler les chapitres de ce sujet

Prepa Help suit le programme de TSI chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Bases de Python : expressions, types, instructions, fonctions
- Récursivité
- Complexité des algorithmes itératifs et récursifs

Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/cnc/tsi