Concours National Commun (CNC), Maroc

Session 2018, filière TSI

Informatique

Ce que traite le sujet

- Partie I : bases de données, arbre généalogique
- Partie II : bioinformatique, recherche d'un motif
- Partie III : calcul numérique, racine n-ième par la méthode de Newton

Copie non officielle du sujet, à partir de la page suivante.

Cette copie est une ressaisie du sujet dans un recueil d'annales, pas le document officiel : elle peut contenir des fautes de frappe. Elle sera remplacée dès que nous aurons une copie officielle.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/cnc/tsi/2018/info

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

CNC 2018 - Filière TSI

Sommaire

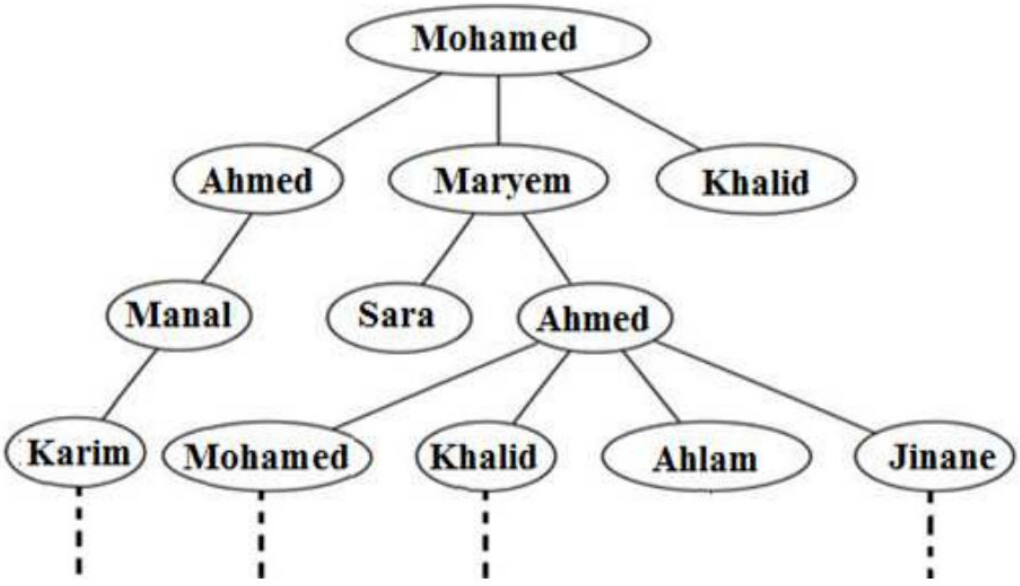
Partie I : Bases de donnée

Arbre généalogique

Dans cette partie, le but est de modéliser une structure qui représente un **arbre généalogique** par une base de données. On commence par représenter l'arbre généalogique sous forme d'un **arbre n-aire descendant** :

En haut de l'arbre se situe l'ancêtre commun à tous les membres de l'arbre généalogique. Et chaque membre de l'arbre est relié vers le bas à chacun de ses enfants. À l'exception de l'ancêtre commun, chaque membre de l'arbre est relié vers le haut à une unique personne : son parent.

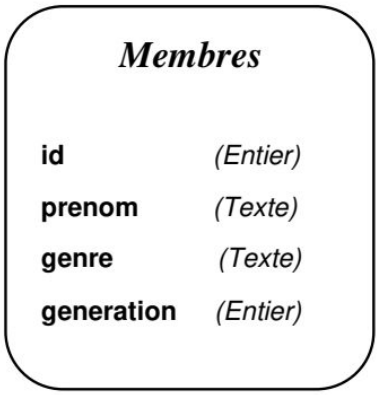
Pour plus de clarté, nous prendrons l'arbre généalogique suivant comme exemple dans toute cette partie :

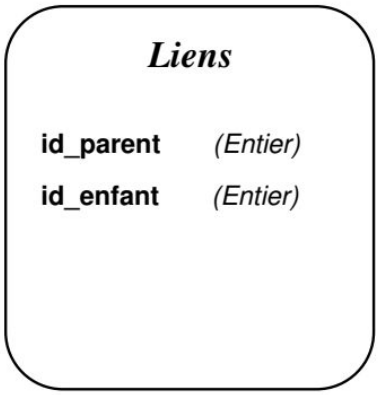


Cet arbre généalogique est composé de plusieurs générations. Le plus grand ancêtre est *'Mohamed'*, il appartient à la génération **0.** Le plus grand ancêtre est appelé : **racine** de l'arbre. Ce dernier possède trois enfants : *'Ahmed'*, *'Maryem'* et *'Khalid'*. Ces trois enfants appartiennent à la génération **1.** *'Khalid'* n'a pas d'enfant, *'Ahmed'* a un enfant, et *'Maryem'* en a deux. Les enfants de *'Ahmed'* et de *'Maryem'* appartiennent à la génération **2**, ... etc. Le nombre total de générations dans l'arbre est appelé : **hauteur** de l'arbre. On peut calculer la hauteur de l'arbre en ajoutant le nombre **1** à la plus grande génération dans l'arbre.

Exemple : La hauteur de l'arbre de l'exemple précédent est **4**.

Dans ce qui suit, on propose de représenter l'arbre généalogique par une base de données composée de deux tables : la table : **membres** et la table : **liens**.





a- Structure de la table **Membres** :

La table **Membres** contient tous les membres de l'arbre généalogique. Elle est composée de **quatre** champs : **id**, **prenom**, **genre** et **generation** :

- ✓ Le champ **id** contient un entier positif unique, qui permet d'identifier, chaque membre dans la table ;
- ✓ Le champ **prenom** contient le prénom d'un membre de l'arbre généalogique ;
- ✓ Le champ **genre** contient le caractère **'M'** pour les masculins, ou **'F'** pour les féminins :
- ✓ Le champ **generation** contient un entier positif qui représente le niveau du membre dans l'arbre généalogique.

b- La structure de la table **Liens** :

La table **Liens** contient tous les liens de parenté parent-enfant entre les membres de l'arbre généalogique. Elle est composée de deux champs : **id_parent** et **id_enfant** :

- ✓ Le champ **id_parent** contient les identifiants des membres parents ;
- ✓ Le champ **id_enfant** contient les identifiants des membres enfants.

c- Exemples d'enregistrements dans les tables **Membres** et **Liens**:

Table : Membres			
id	**prenom**	**genre**	**generation**
1	Mohamed	M	0
2	Ahmed	M	1
3	Maryem	F	1
4	Khalid	M	1
5	Manal	F	2
6	Sara	F	2
7	Ahmed	M	2
8	Jinane	F	3
…	**…**	**…**	**…**

Table : Liens	
id_parent	**id_enfant**
1	**2**
1	**3**
1	**4**
2	**5**
3	**6**
3	**7**
7	**8**
…	**…**

Q.1- Déterminer la *clé primaire* de la table **Membres**, et la *clé primaire* de la table **Liens**.

Justifier votre réponse.

Q.2- Écrire une requête SQL, qui donne pour résultat **le prénom et le genre du plus grand ancêtre des membres.**

Q.3- Écrire une requête SQL, qui donne pour résultat **les prénoms et les générations, sans répétition, des membres masculins, dont le prénom commence par 'A', et se termine par 'D', et contient 'E'.**

Q.4- Écrire une requête SQL, qui donne pour résultat **la hauteur de l'arbre.**

Q.5- Écrire une requête SQL, qui donne pour résultat **les prénoms et les genres des enfants du parent dont l'id_parent = 15, triés dans l'ordre alphabétique des prénoms.**

Q.6- Écrire une requête SQL qui donne pour résultat **les identifiants, les prénoms et les générations des membres ayant un nombre d'enfants supérieur ou égal à 3, triés dans l'ordre décroissant du nombre d'enfants.**

Q.7- Écrire une requête SQL qui **crée une nouvelle table nommée Membres_1, ayant la même structure que la table Membres.**

Q.8- Écrire une requête SQL qui **copie dans la table Membres_1, tous les membres de la table Membres, qui possèdent le même prénom que celui du plus grand ancêtre.**

-

Partie II :

Bioinformatique : Recherche d'un motif

La bioinformatique, c'est quoi?

Au cours des trente dernières années, la récolte de données en biologie a connu un boom quantitatif grâce notamment au développement de nouveaux moyens techniques servant à comprendre l'ADN et d'autres composants d'organismes vivants. Pour analyser ces données, plus nombreuses et plus complexes aussi, les scientifiques se sont tournés vers les nouvelles technologies de l'information. L'immense capacité de stockage et d'analyse des données qu'offre l'informatique leur a permis de gagner en puissance pour leurs recherches. La bioinformatique sert donc à stocker, traiter et analyser de grandes quantités de données de biologie. Le but est de mieux comprendre et mieux connaître les phénomènes et processus biologiques.

Dans le domaine de la bioinformatique, la recherche de sous-chaîne de caractères, appelée **motif**, dans un texte de taille importante, était un composant important dans beaucoup d'algorithmes. Puisqu'on travaille sur des textes de tailles très grandes, on a toujours cette obsession de l'efficacité des algorithmes : moins on a à faire de comparaisons, plus rapide sera l'exécution des algorithmes.

Le motif à rechercher, ainsi que le texte dans lequel on effectue la recherche, sont écrits dans un alphabet composé de quatre caractères : **'A'** , **'C'** , **'G'** et **'T'**. Ces caractères représentent les quatre bases de l'ADN : **A**dénine, **C**ytosine, **G**uanine et **T**hymine.

Codage des caractères de l'alphabet

L'alphabet est composé de quatre caractères seulement : **'A'**, **'C'**, **'G'** et **'T'.** Dans le but d'économiser la mémoire, on peut utiliser un codage binaire *réduit* pour ces quatre caractères.

Q. 1 : Quel est le *nombre minimum de bits* nécessaires pour coder quatre éléments ?

Q. 2 : Donner un exemple de code binaire pour chaque caractère de cet alphabet.

Préfixe d'une chaîne de caractères

*Un **préfixe** d'une chaîne de caractères **S** non vide, est une sous-chaîne non vide de **S**, composée de la suite des caractères entre la première position de **S** et une certaine position dans **S**.*

Exemple : S = 'TATCTAGCTA'

- ✓ La chaîne **'TAT'** est un préfixes de **'TAT**CTAGCTA' ;
- ✓ La chaîne **'TATCTA'** est un préfixes de **'TATCTA**GCTA' ;
- ✓ La chaîne **'TATCTAGCTA'** est un préfixes de **'TATCTAGCTA'** ;
- ✓ La chaîne **'T'** est un préfixes de **'T**ATCTAGCTA' ;
- ✓ La chaîne 'CTAGC' n'est pas un préfixe de S.

Q. 3 : Écrire une fonction **prefixe (M, S)**, qui reçoit en paramètres deux chaînes de caractères **M** et **S** non vides, et qui retourne **True** si la chaîne **M** est un préfixe de **S**, sinon, elle retourne **False**.

Q. 4 : Quel est le nombre de *comparaisons élémentaires* effectuées par la fonction **prefixe (M, S)**, dans le pire cas ? Déduire la complexité de la fonction **prefixe (M, S)** dans le pire cas.

Suffixe d'une chaîne de caractères

*Un **suffixe** d'une chaîne de caractères **S** non vide, est une sous-chaîne non vide de **S**, composée de la suite des caractères, en partant d'une certaine position **p** dans **S**, jusqu'à la dernière position de **S**. L'entier **p** est appelée : position du suffixe.*

Exemple : S = 'TATCTAGCTA'

- ✓ La chaîne **'CTAGCTA'** est un suffixe de 'TAT**CTAGCTA'**, sa position est : **3** ;
- ✓ La chaîne **'GCTA'** est un suffixe de 'TATCTA**GCTA'**, sa position est : **6** ;
- ✓ La chaîne 'TATCTAGCTA' est un suffixe de **'TATCTAGCTA'**, sa position est : **0** ;
- ✓ La chaîne 'CTAGC' n'est pas un suffixe de S.

Q. 5 : Écrire une fonction **liste_suffixes (S)**, qui reçoit en paramètre une chaîne de caractères **S** non vide, et qui renvoie une liste de tuples. Chaque tuple de cette liste est composé de deux éléments : un suffixe de **S**, et la position **p** de ce suffixe dans **S** ($0 \leq p <$ taille(S)). Les tuples de cette liste sont ordonnés dans l'ordre croissant des positions des suffixes.

Exemple : S = 'TATCTAGCTA'

La fonction **liste_suffixes (S)** renvoie la liste :

[('TATCTAGCTA', **0**), ('ATCTAGCTA', **1**), ('TCTAGCTA', **2**), ('CTAGCTA', **3**), ('TAGCTA', **4**), ('AGCTA', **5**), ('GCTA', **6**), ('CTA', **7**), ('TA', **8**), ('A', **9**)]

Recherche séquentielle d'un motif

La **recherche séquentielle**, d'un motif **M** dans une chaîne de caractères **S** non vide, consiste à parcourir la liste **L** des suffixes de **S** (voir Question. 5), en cherchant si le motif **M** est un préfixe du suffixe d'un tuple de la liste **L**. Si oui, la fonction retourne la position de ce suffixe.

*Le parcours, de la liste L, ne doit considérer que les positions possibles pour le motif **M**, c'est-à-dire les positions **p** tels que : **0 ≤ p ≤ (taille de L – taille de M)**.*

Q. 6 : Écrire une fonction : **recherche_sequentielle (M, L)**, qui, en utilisant le principe de *la recherche séquentielle* dans **L**, renvoie la position du premier tuple, tel que **M** est un préfixe du suffixe de ce tuple, s'il existe ; sinon, la fonction renvoie **None**.

NB : Dans cette fonction, on ne doit pas utiliser la méthode index().

Exemples : La liste des suffixes de la chaîne 'TATCTAGCTA' est :

L = [('TATCTAGCTA', **0**), ('ATCTAGCTA', **1**), ('TCTAGCTA', **2**), ('CTAGCTA', **3**), ('TAGCTA', **4**), ('AGCTA', **5**), ('GCTA', **6**), ('CTA', **7**), ('TA', **8**), ('A', **9**)]

- ✓ **recherche_sequentielle** ('CTA', L) renvoie la position **3** ; *'**CTA**' est préfixe de '**CTA**GCTA'*
- ✓ **recherche_sequentielle** ('TA', L) renvoie la position **0** ; *'**TA**' est préfixe de '**TA**TCTAGCTA'*
- ✓ **recherche_sequentielle** ('CAGTA', L) renvoie **None**. *'CAGT' n'est préfixe d'aucun suffixe*

Q. 7 : On pose **m** et **k** les tailles respectives de la chaîne **M** et la liste **L**. Déterminer, en fonction de **m** et **k**, la complexité de la fonction recherche_sequentielle (M, L), dans le pire cas. Et, donner un exemple, de chaînes **M** et **S**, qui atteigne cette complexité.

Tri de la liste des suffixes

Q. 8 : Écrire une fonction **tri_liste (L)**, qui reçoit en paramètre la liste **L** des suffixes d'une chaîne de caractères non vide, créée selon le principe décrit dans la question **5**. La fonction **tri_liste**, trie les éléments de **L** dans l'ordre alphabétique des suffixes. (Voir exemple suivant)

NB : On ne doit pas utiliser la fonction sorted(), ou la méthode sort().

Exemple :

L = [('TATCTAGCTA', **0**), ('ATCTAGCTA', **1**), ('TCTAGCTA', **2**), ('CTAGCTA', **3**), ('TAGCTA', **4**), ('AGCTA', **5**), ('GCTA', **6**), ('CTA', **7**), ('TA', **8**), ('A', **9**)]

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

[(**'A'**, 9) , (**'AGCTA'**, 5) , (**'ATCTAGCTA'**, 1) , (**'CTA'**, 7) , (**'CTAGCTA'**, 3) , (**'GCTA'**, 6) , (**'TA'**, 8) , (**'TAGCTA'**, 4) , (**'TATCTAGCTA'**, 0) , (**'TCTAGCTA'**, 2)]

Recherche dichotomique d'un motif :

La liste des suffixes **L** contient les tuples composés des suffixes d'une chaîne de caractères **S** non vide, et de leurs positions dans **S**. Si la liste **L** est triée dans l'ordre alphabétique des suffixes, alors, la recherche d'un motif **M** dans **S**, peut être réalisée par une simple *recherche dichotomique* dans la liste **L**.

Q. 9 : Écrire une fonction **recherche_dichotomique (M, L)**, qui utilise le *principe de la recherche dichotomique*, pour rechercher le motif **M** dans la liste des suffixes **L** triée dans l'ordre alphabétique des suffixes : Si **M** est un préfixe d'un suffixe dans un tuple de **L**, alors la fonction retourne la position de ce suffixe. Si la fonction ne trouve pas le motif **M** recherché, alors la fonction retourne **None**.

Exemple :

La liste des suffixes de **'TATCTAGCTA'**, triée dans l'ordre alphabétique des suffixes est :

L = [('A', 9) , ('AGCTA', 5) , ('ATCTAGCTA', 1) , ('CTA', 7) , ('CTAGCTA', 3) , ('GCTA', 6) , ('TA', 8) , ('TAGCTA', 4) , ('TATCTAGCTA', 0) , ('TCTAGCTA', 2)]

- ✓ **recherche_dichotomique ('CTA', L)** renvoie la position **3** ; *'**CTA**' est préfixe de '**CTA**GCTA'*
- ✓ **recherche_dichotomique ('TA', L)** renvoie la position **0** ; *'**TA**' est un préfixe de '**TA**TCTAGCTA'*
- ✓ **recherche_dichotomique ('CAGT', L)** renvoie **None**. *'CAGT' n'est préfixe d'aucun suffixe*

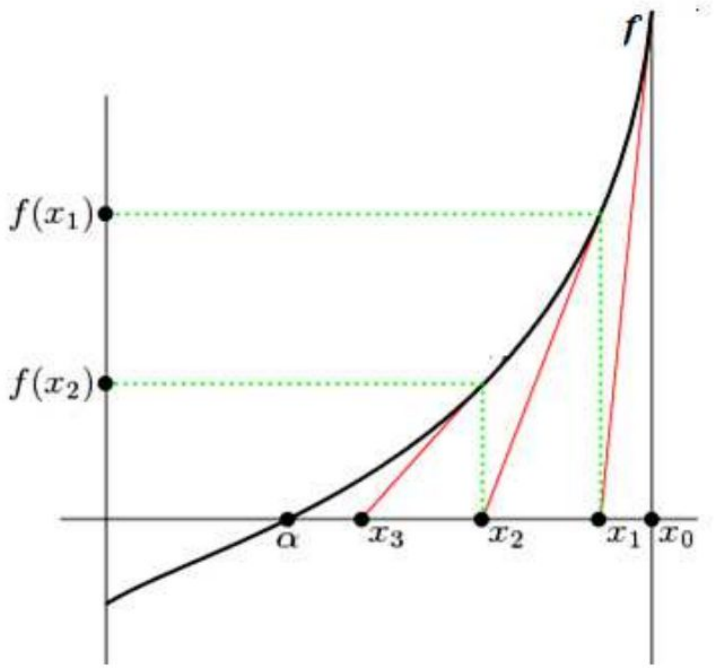
Q. 10 : On pose **m** et **k** les tailles respectives de la chaîne **M** et la liste **L**. Déterminer, en fonction de **m** et **k**, la complexité de la fonction **recherche_dichotomique (M, L)**, dans le pire cas. Justifier votre réponse.

-

Partie III : Calcul numérique

Calcule de la racine $n^{ième}$ par la méthode de 'Newton'

De nombreux problèmes d'économie, de mathématiques ou de physique se concluent par la résolution d'une équation $f(x) = 0$. Bien souvent, il n'est pas possible de résoudre exactement cette équation, et on cherche une valeur approchée de la solution (ou des solutions). **Newton** a proposé une méthode générale pour obtenir une telle approximation. L'idée est de remplacer la courbe représentative de la fonction par sa tangente.



On part d'un point x_0 de l'intervalle de définition de f, et on considère la tangente à la courbe représentative de f en $(x_0 , f(x_0))$. On suppose que cette tangente coupe l'axe des abscisses (c.à.d. $f'(x_0)$ non nul).

Soit x_1 l'abscisse de l'intersection de la tangente avec l'axe des abscisses : $\boldsymbol{x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}}$

Puisque la tangente est proche de la courbe, on peut espérer que x_1 donne une meilleure estimation d'une solution de l'équation $f(x) = 0$ que x_0.

On recommence alors le procédé à partir de x_1, on obtient x_2 : $\boldsymbol{x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}}$

…

Ainsi, on construit par récurrence une suite (x_n) définie par : $\boldsymbol{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}}$

Sous de bonnes hypothèses sur f, assez restrictives, (x_n) converge vers la solution de l'équation $f(x) = 0$, et la convergence est très rapide.

A- Approximation de la dérivée par la moyenne des taux d'accroissement à gauche et à droite :

Si f est une fonction dérivable en x_i, on peut toutefois obtenir une approximation de $f'(x_i)$ par la moyenne des taux d'accroissement à gauche et à droite de x_i :

Pour un réel $h > 0$ (très proche de **0**) :

$$f'(x_i) \approx \frac{f(x_i + h) - f(x_i - h)}{2h}$$

Q.1- Montrer que : $x_{n+1} = x_n - 2h\frac{f(x_n)}{f(x_n+h) - f(x_n-h)}$

Q.2- Écrire une fonction : **newton (*f*, x0, eps, h)** qui reçoit en paramètres :

- La fonction f ;
- Le premier terme x_0 de la suite (x_n) ;
- Le réel h strictement positif très proche de **0** ;
- Le réel ***eps*** strictement positif qui représente la précision.

La fonction renvoie le premier terme x_k de la suite (x_n) tel que : $|x_k - x_{k-1}| \leq eps$

B- Calcul de la racine n-ème positive d'un réel positif

Pour calculer la racine n-ème positive d'un réel positif a, il suffit de trouver la solution positive de l'équation : $x^n - a = 0$, par la méthode de Newton.

Dans ce qui suit, on suppose que a et n sont deux variables globales déclarées et initialisées :

a est un réel positif, et n un entier strictement positif.

Q. 3- Définir la fonction **g (x)**, qui reçoit en paramètre un réel positif x et qui renvoie : $x^n - a$.

Q. 4- On suppose que les modules suivant sont importé :

from numpy import *

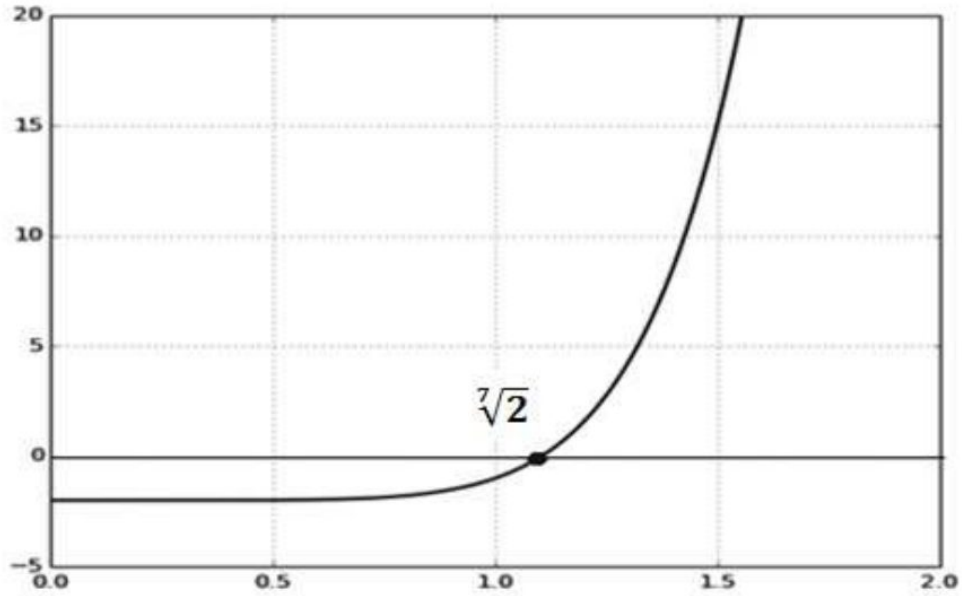
from matplotlib.pyplot import *

Écrire le code du programme Python qui permet de tracer la courbe de la fonction g, dans l'intervalle **[0 , a]**.

Le nombre de points générés dans la courbe est : **500**

Exemple :

Pour **a=2.0** et **n=7**, on obtient la courbe suivante :



Représentation graphique de la fonction : $g(x) = x^7 - 2$

Q. 5- Écrire le code du programme Python qui utilise la fonction **newton** (), et qui affiche la valeur approximative de la racine n-ème positive de a : $\sqrt[n]{a}$, avec la précision : 10^{-12} et $h = 0.00001$.

Exemple :

Pour **n=7** et **a=2.0**, le programme affiche la racine **7ème** positive de **2** : **1.1040895136738123**

~ ~ ~ ~ ~ ~ **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
- Complexité des algorithmes itératifs et récursifs
- Tri par fusion et tri rapide
- Architecture de l'ordinateur et représentation des nombres
- Suites récurrentes
- Dérivabilité et convexité

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