TS spé

Exercices sur les algorithmes liés à la divisibilité et à la division euclidienne

1 1°) Écrire une fonction Python `nbrespairs(n)` qui prend pour argument un entier naturel n supérieur ou égal à 1 et qui renvoie les n premiers entiers naturels pairs dans l'ordre croissant.
Proposer une version qui renvoie la réponse sous forme de liste.
2°) Écrire une fonction Python `nbresimpairs(n)` qui prend pour argument un entier naturel n supérieur ou égal à 1 et qui renvoie les n premiers entiers naturels impairs dans l'ordre croissant.
Proposer une version qui renvoie la réponse sous forme de liste.

2 1°) Écrire une fonction Python `pairsentre2bornes(a,b)` qui prend pour arguments deux entiers relatifs a et b tels que $a \leqslant b$ et qui renvoie la liste de tous les entiers pairs entre a et b dans l'ordre croissant.
2°) Même question avec une fonction `impairsentre2bornes(a,b)`.

3 Écrire une fonction Python `pairs(n)` qui prend pour argument un entier naturel n et qui renvoie la liste des entiers naturels pairs inférieurs ou égaux à n.
Idem pour une fonction Python `impairs(n)`.

4 Écrire une fonction Python `convert(n,b)` qui convertit un entier naturel n (donné par son écriture en base dix) en base b où b est un entier naturel supérieur ou égal à 2.
On rappelle que la calculatrice Numworks fait toute seule les conversions en binaire.

5 On considère la fonction suivante ex(n,a) qui prend pour arguments un entier naturel n et un entier naturel a supérieur ou égal à 1.

```
Fonction ex(n,a)
    Tantque n ⩾ a Faire
        n ← n − a
    FinTantque
    Renvoyer n
FinFonction
```

Recopier cette fonction dans un cadre (sur une même page, on ne doit pas avoir à passer sur une nouvelle page).
1°) On prend $a = 11$.
Déterminer les valeurs de ex(35,11), ex(50,11), ex(55,11), ex(5,11).
On fera tourner la boucle « à la main » pour les valeurs de n suivantes saisies en entrée :
$n = 35$; $n = 50$; $n = 55$.

Pour chaque valeur de n, on fera un tableau d'évolution de la variable n sur le modèle ci-dessous.

Étape	**0**	**1**	**2**	
Condition $n \geqslant 11$		vraie	vraie	
Valeur de n	35	24	13	

Deux remarques importantes :

- Il est inutile de programmer l'algorithme sur calculatrice.
- Pour chaque valeur de n saisie en entrée, faire une phrase réponse sur le modèle suivant (à recopier et compléter) : « Pour $n = ...$ saisi en entrée, la valeur de u affichée en sortie est … ».

Pour un entier naturel n quelconque, quel est le nombre entier naturel fourni par cet algorithme en sortie ?
2°) De manière générale, quel est le rôle de la fonction ex(n,a) ?
3°) Réaliser le programme correspondant en Python et vérifier les résultats du 1°).

6 Années bissextiles

On dit qu'une année de millésime a est *bissextile* si l'on est dans l'un des deux cas suivants :
cas 1 : a est divisible par 4 mais pas par 100 ;
cas 2 : a est divisible par 400.
1°) Déterminer celles des années 2020, 2034, 2100, 2000, 1900 qui sont bissextiles.
2°) Écrire une fonction Python `bissext(a)` qui teste si une année de millésime a est bissextile ou non.
3°) Écrire une fonction Python `list_bissext(a,b)` qui prend pour arguments deux années a et b et qui renvoie la liste des années bissextiles comprises entre les années a et b incluses.
À l'aide de cette fonction, donner la liste des années bissextiles entre 2000 et 2100.

Une année (qui correspond à la durée de révolution de la Terre autour du Soleil) dure 365,2422 jours. Afin que notre calendrier coïncide avec cette période, certaines années durent 365 jours et d'autres 366 jours.
La partie décimale de 365,2422 valant presque un quart, on ajoute tous les 4 ans un jour à notre calendrier (le 29 février).

Le calendrier julien est introduit par Jules César en 46 av. J.-C. pour remplacer le calendrier romain. Il est utilisé dans la Rome antique à partir de 45 av. J.-C.. Il reste employé en Europe jusqu'à son remplacement par le calendrier grégorien à la fin du XVI^{e} siècle.

Le calendrier grégorien est un calendrier solaire conçu à la fin du XVI^{e} siècle pour corriger la dérive séculaire du calendrier julien alors en usage. Il porte le nom de son instigateur, le pape Grégoire XIII. Adopté à partir de 1582 dans les États catholiques, puis dans les pays protestants, son usage s'est progressivement étendu à l'ensemble du monde au début du XX^{e} siècle. Le calendrier grégorien s'est imposé dans la majeure partie du monde pour les usages civils ; de nombreux autres calendriers sont utilisés pour les usages religieux ou traditionnels.

Les années comptent 365 jours dont 28 en février sauf les années bissextiles à 366 jours dont 29 en février.
Les **années bissextiles** sont :
- les années dont le millésime est multiple de 4 mais pas de 100
- et les années dont le millésime est multiple de 2000.

Ces décisions ont été prises peu à peu au cours des siècles pour ajuster notre calendrier à l'année des saisons (365,2422 jours). Une année de 365 jours, c'est simple mais trop court ! Il faut donc rajouter des jours. Le calendrier julien, fondé par Jules César, comporte un jour doublé tous les quatre ans. Il s'agit du sixième jour avant les calendes de mars qui devient donc *bis-sextus martias* d'où le nom d'année bissextile.
Avec une année de 366 jours tous les quatre ans, la durée moyenne de l'année est de 365,25 jours. Pas mal par rapport à l'année des saisons, mais trop long...
Au bout de quatre siècles, le décalage est de trois jours. En 1582, il atteint 12 jours ce qui pose des problèmes aux autorités ecclésiastiques pour fixer certaines fêtes religieuses comme celle de Pâques. Le pape Grégoire XIII ordonne alors une réforme du calendrier : le calendrier grégorien est né. Tout d'abord, il faut résorber le décalage constaté : 10 jours sont simplement supprimés en 1582. Ensuite, il faut éliminer les causes du décalage et donc réduire le nombre d'années bissextiles : les années séculaires deviennent non bissextiles sauf si leur millésime est multiple de 400. L'année grégorienne ne comporte plus en moyenne que 365,2425 jours. Elle est encore trop longue de …0,0003 jours par rapport à l'année des saisons. Dans 10 000 ans, notre calendrier aura trois jours de trop !

Pour les exercices **7** et **8**, on rappelle l'algorithme $(\mathcal{A})$ suivant qui, pour un entier naturel $n \geqslant 1$ saisi en entrée, affiche ses diviseurs positifs en sortie.

Entrée :
Saisir n

Traitement et sorties :
Pour i allant de 1 à n **Faire**
  Si $i \mid n$
    Alors afficher i
  FinSi
FinPour

On pourra utiliser une version « liste » de cet algorithme.

7 Pour tout entier naturel $n \geqslant 1$, on note $\mathrm{d}(n)$ le nombre de ses diviseurs positifs.
1°) Recopier et compléter le tableau de valeurs suivant :

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\mathrm{d}(n)$																

2°) On admet que l'on ne peut pas donner une expression de $\mathrm{d}(n)$ en fonction de n.

On a juste $1 \leqslant \mathrm{d}(n) \leqslant n$ pour tout entier naturel $n \geqslant 1$ et même $2 \leqslant \mathrm{d}(n) \leqslant n$ pour tout entier naturel $n \geqslant 2$.

Pour calculer la valeur de $\mathrm{d}(n)$ pour un entier naturel $n \geqslant 1$ fixé, on peut utiliser une fonction.
En s'inspirant de l'algorithme $(\mathcal{A})$, écrire une fonction Python `d(n)` qui prend pour argument un entier $n \geqslant 1$ et qui renvoie en sortie la valeur de $\mathrm{d}(n)$.
Programmer cette fonction et déterminer :
- la valeur de $\mathrm{d}(1000)$;
- le plus petit entier naturel qui admet exactement 15 diviseurs positifs.

Proposer une autre façon d'écrire la fonction `d(n)` en utilisant les listes.

8 Pour tout entier naturel $n \geqslant 1$, on note $\mathrm{S}(n)$ la somme de ses diviseurs positifs.
On admettra que l'on ne peut pas donner une expression de $\mathrm{S}(n)$ en fonction de n.
1°) En s'inspirant de l'algorithme $(\mathcal{A})$, écrire une fonction Python `S(n)` prenant pour argument un entier $n \geqslant 1$ et qui renvoie en sortie la valeur de $\mathrm{S}(n)$. Programmer cette fonction.

Réaliser le programme correspondant sur calculatrice.

Proposer une autre façon d'écrire la fonction `S(n)` en utilisant les listes.

8bis Nombres abondants, déficients, parfaits

①

• Exemple :

Les diviseurs positifs de 12 sont 1, 2, 3, 4, 6, 12.

La somme des diviseurs positifs stricts de 12 est 16. Comme 16 est strictement supérieur à 12, on dit que le nombre est abondant.

• Définition générale :

On dit qu'un entier naturel *n* supérieur ou égal à 2 est *abondant* lorsque la somme de ses diviseurs positifs stricts est strictement supérieure à *n*.

②

• Exemple :

Les diviseurs positifs de 10 sont 1, 2, 5, 10.

La somme des diviseurs positifs stricts de 10 est 8. Comme 8 est strictement inférieur à 10, on dit que le nombre 10 est déficient.

• Définition générale :

On dit qu'un entier naturel *n* supérieur ou égal à 2 est *déficient* lorsque la somme de ses diviseurs positifs stricts est strictement inférieure à *n*.

③

• Exemple :

Les diviseurs positifs de 28 sont 1, 2, 4, 7, 14, 28.
La somme des diviseurs positifs stricts de 28 est 28. Comme la somme est égale à 28, on dit que le nombre 28 est parfait.

• Définition générale :

On dit qu'un entier naturel *n* supérieur ou égal à 2 est *parfait* lorsque la somme de ses diviseurs positifs stricts est égale à *n*.

On avait déjà rencontré les nombres parfaits dans l'un des exercices sur « Multiples et diviseurs ».

Donner un exemple de nombre abondant, déficient, parfait.

Écrire une fonction somdivstric(n) qui prend pour argument un entier naturel *n* non nul et qui renvoie la liste des diviseurs positifs autres que *n*.

Écrire une fonction Python `typenb(n)` qui détecte si un entier naturel $n \geqslant 2$ est abondant, déficient ou parfait. On utilisera la fonction `S(n)` définie dans l'exercice **8** .

Les nombres parfaits abondants, déficients sont connus depuis l'Antiquité. Euclide en parle dans les *Éléments*.

Voir article Wikipedia nombres presque parfaits.
Expliquer pourquoi 64 n'est pas un nombre parfait, à une unité près.
On dit que 64 est presque parfait.
3) Déterminer tous les nombres presque parfaits inférieurs à 20.

9 Chiffrement de Hill

Le but de l'exercice est d'étudier une méthode de chiffrement publiée en 1929 par le mathématicien et cryptologue américain Lester Hill. C'est un chiffrement polygraphique, c'est-à-dire qu'on ne (dé)chiffre pas les lettres les unes après les autres, mais par paquets. Nous étudierons la version bigraphique, c'est-à-dire que nous grouperons les lettres deux par deux, mais on peut imaginer des paquets plus grands.

On veut coder un mot de deux lettres selon la procédure suivante :

Étape 1 :

Chaque lettre du mot est remplacée par un entier en utilisant le tableau ci-dessous :

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

On obtient un couple d'entiers $(x_1 ; x_2)$ où x_1 correspond à la première lettre du mot et x_2 correspond à la deuxième lettre du mot.

Étape 2 :

$(x_1 ; x_2)$ est transformé en $(y_1 ; y_2)$ tel que y_1 est le reste de la division euclidienne de $11x_1 + 3x_2$ par 26 et y_2 est le reste de la division euclidienne $7x_1 + 4x_2$ par 26.

Étape 3 :

$(y_1 ; y_2)$ est transformé en un mot de deux lettres en utilisant le tableau de correspondance donné dans l'étape 1.

Exemple :

$$\underset{\text{mot en clair}}{\text{TE}} \xrightarrow{\text{étape 1}} (19\,;4) \xrightarrow{\text{étape 2}} (13\,;19) \xrightarrow{\text{étape 3}} \underset{\text{mot codé}}{\text{NT}}$$

Écrire une fonction Python qui prend pour argument un groupe de 2 lettres représentés par deux entiers entre 0 et 25 et qui renvoie les nombres correspondants aux lettres codées par le chiffrement.

10 On considère la fonction Python `nbc(n)` dans l'encadré ci-dessous qui prend pour argument un entier naturel *n* :

```
def nbc(n):
    x = 1
    while n>= 10:
          n = n // 10
          x = x+1
    return x
```

1°) Recopier cette fonction.
Faire tourner cette fonction « à la main » pour $n = 407$. Recommencer pour plusieurs valeurs de *n*.
Que renvoie cette fonction ?
2°) Programmer cette fonction sur calculatrice.

11 (Reprise de l'exercice 17 de feuille d'exercices sur la division euclidienne)
Le numéro INSEE est composé de 13 chiffres (matricule), suivi d'une clé de contrôle de 2 chiffres. Pour déterminer la clé C, on calcule le reste de la division euclidienne par 97 du numéro N formé des 13 premiers chiffres, alors $C = 97 - R$.
Attention, la clé de contrôle s'écrit avec deux chiffres.
Par exemple, si $R = 91$, alors $C = 6$. La clé de contrôle s'écrit 06.
Rappel : Le premier chiffre du numéro INSEE indique le sexe, 1 pour sexe masculin et 2 pour le sexe féminin.
1°) Écrire une fonction Python `cle(m)` qui prend en argument le matricule m (nombre composé de treize chiffres) et qui retourne la clé C.
2°) Écrire une fonction Python `controle(n)` qui prend en argument une liste composée des quinze chiffres du numéro INSEE et renvoie True si la clé est correcte, False sinon.

12 1°) Soit *n* un entier naturel qui s'écrit avec deux chiffres en base dix.
On pose $n = \overline{ab}$.
Recopier et compléter les phrases suivantes :
a est le … de la division euclidienne de *n* par 10.
b est le … de la division euclidienne de *n* par 10.
2°) On appelle retourné d'un entier naturel le nombre dont les chiffres sont ceux de l'entier naturel considéré écrits dans l'autre sens. Par exemple, le retourné de 25 est 52, le retourné de 361 est 163.
Écrire une fonction Python `retourne(n)` qui prend pour argument un entier naturel *n* qui s'écrit en base dix avec deux chiffres et qui renvoie son retourné.
Indication : Utiliser les résultats du 1°).

13 On considère la fonction Python `retourne(x)` qui prend pour argument un entier naturel *x*.

```
def retourne(x) :
    y = 0
    while x != 0:
          y = (y * 10) + (x % 10)
          x = x // 10
    return y
```

1°) Faire tourner « à la main » le programme pour $x = 347$.
Que renvoie la fonction `retourne(x)` ?
3°) Programmer cette fonction.

14 **Générateur d'entiers pseudo-aléatoires**
On considère la suite (u_n) d'entiers naturels définie sur $\mathbb{N}$ par son premier terme $u_0 = 100$ et par la relation de récurrence u_{n+1} = reste de la division euclidienne de $15091 \times u_n$ par 64007 pour tout entier naturel *n*.
Programmer le calcul des termes de la suite (u_n) en Python.
Que valent u_{30}, u_{300}, u_{3000} ?
Recommencer en prenant d'autres valeurs de u_0 dans $\mathbb{N}$.

Il s'agit d'une méthode utilisée en informatique pour obtenir des nombres pseudo-aléatoires (pour produire des nombres aléatoires) depuis qu'elle a été inventée en 1948 par D. H Lehmer (générateur congruentiel linéaire).
Une autre méthode est la méthode des carrés médians de Von Neumann en 1946.

15 Dans cet exercice, on s'intéresse à la somme des chiffres qui composent l'écriture en base dix d'un entier naturel.
Par exemple, pour l'entier naturel qui s'écrit 450034 en base dix, on calcule $4+5+0+0+3+4=16$.
La somme des chiffres est égale à 16.
Écrire une fonction Python `SommeChiffres(n)` qui prend pour argument un entier naturel *n* et qui renvoie la somme de ses chiffres de son écriture en base dix.
Indication : Utiliser des divisions euclidiennes par 10.
Recommencer avec le produit des chiffres.

16 1°) Le nombre 153 possède une propriété remarquable : il est égal à la somme des cubes de ses chiffres dans son écriture en base dix. En effet, on a : $1^3+5^3+3^3 = 1+125+27 = 153$.
À l'aide d'un programme Python, déterminer les entiers naturels à 3 chiffres qui possèdent la même propriété.
On ne tiendra pas compte des nombres comme 001 qui commencent par 0.
2°) Le nombre 4150 possède une propriété remarquable : il est égal à la somme des puissances cinquièmes des chiffres de on écriture en base dix. En effet, on a : $4150 = 4^5+1^5+5^5+0^5$.
En raisonnant par l'absurde, démontrer que si un entier naturel vérifie cette propriété, il a au plus 6 chiffres.
En déduire que seul un nombre fini d'entiers vérifie cette propriété.
Écrire un programme Python qui renvoie la liste des nombres satisfaisant cette propriété.

17 Écrire une fonction Python qui affiche le premier chiffre de l'écriture en base dix d'un entier naturel.

18 Écrire une fonction Python d'en-tête `def suppression(L):` qui prend pour argument une liste L d'entiers relatifs et qui renvoie la liste constituée des éléments non divisibles par 5.

19 Écrire une fonction Python d'en-tête def `racines_empilées(L):` qui prend en argument une liste $L = [a_0, a_1, \ldots, a_n]$ de réels positifs ou nuls et qui renvoie la valeur de $\sqrt{a_n + \sqrt{a_{n-1} + \ldots + \sqrt{a_0}}}$.
Tester la fonction avec une liste composée que de 0 ou que de 1.

20 Écrire une fonction Python d'en-tête def `fractions_empilées(L):` qui prend en argument une liste $L = [a_0, a_1, \ldots, a_n]$ de réels positifs ou nuls et qui renvoie la valeur de $\cfrac{1}{a_n + \cfrac{1}{\ldots\ldots \\ a_1 + \cfrac{1}{a_0}}}$.

21 On rappelle la méthode suivante l'écriture en base deux d'un entier naturel (écriture binaire).

Exemple : Déterminons l'écriture en base deux du nombre qui s'écrit 25 en base dix.

On utilise un « algorithme glouton ».

On cherche la plus grande puissance de 2 inférieure ou égale à 25. Il s'agit de $16 = 2^4$.
On écrit $25 = 16 + 9$.

On recommence avec 9.
On cherche la plus grande puissance de 2 inférieure ou égale à 9. Il s'agit de $8 = 2^3$.
On écrit : $25 = 16 + 8 + 1$.

On s'arrête là. On écrit 25 comme somme de puissances de 2.
On a : $25 = 2^4 + 2^3 + 2^0$.
On complète avec les autres puissances de 2 et des coefficients 0 ou 1.
On peut faire la réécriture suivante : $25 = 1\times 2^4 + 1\times 2^3 + 0\times 2^2 + 0\times 2^1 + 1\times 2^0$.

Cette dernière égalité donne la décomposition en base deux de 25. On peut écrire : $25 = \overline{11001}^{(2)}$.

Écrire une fonction Python qui prend en argument un entier naturel *n* et qui renvoie son écriture en base deux en utilisant la plus grande puissance de 2 inférieure ou égale à cet entier naturel.

22 Écrire une fonction Python qui prend en argument un entier naturel *n* supérieur ou égal à 1 et qui renvoie la fraction d'entiers naturels non nuls inférieurs ou égaux à *n* la plus proche de π.
Donner les résultats obtenus pour $n = 100$ et $n = 1000$.
Recommencer pour ln 2.

23 Écrire une fonction Python d'en-tête `def nbr_pairs(L):` qui prend pour argument une liste L d'entiers relatifs et qui renvoie le nombre de nombre pairs de la liste.

24 Écrire une fonction Python d'en-tête `def nbr_posstrict(L):` qui prend pour argument une liste L de réels et qui renvoie le nombre de réels strictement positifs.

Corrigé

Dans un certain nombre d'exercices, on peut utiliser la notion de fonction en programmation.

Le 12 décembre 2022

T exp

Écriture générique d'un nombre pair : $2k$ avec *k* entier relatif
Écriture générique d'un nombre impair : $2k+1$ avec *k* entier relatif

On va créer une liste vide à laquelle on va rajouter des éléments.

On utilise la fonction append pour rajouter des éléments. Syntaxe : L.append(élément à rajouter)

```
L=[]
for i range(n) :
      L.append(2*i)
return L
```

La dernière valeur de *i* est $n-1$.
Le dernier entier pair affiché sera $2(n-1)$.

Les *n* premiers entiers pairs sont 0, 2, …, $2(n-1)$: il s'agit des entiers de la forme $2k$ pour *k* compris entre 0 et n-1.
Pour $n = 1$ $L = [0]$ La liste du premier entier naturel pair.
Pour $n = 2$ $L = [0, 2]$ La liste des deux premiers entiers naturels pairs.

`range(n)` correspond à l'intervalle d'entiers $[\![0, n-1]\!]$
exemple : `range(5)` : intervalle d'entiers $[\![0, 4]\!]$ composé des nombres 0, 1, 2, 3, 4.

1

1°) Les n premiers entiers pairs sont $0=2\times0$, $2=2\times1$, …, $2(n-1)=2\times(n-1)$.

1ère idée : utilisation d'une boucle for

```
def nbrespairs(n):
    L=[]
    for k in range(n):
        L.append(2*k)
    return L
```

`L.append(2*k)` ou `L+[2*k]`
`range(0,n)` ou `range(n)`

```
def nbrespairs(n):
    L=[2*k for k in range(n)]
    return L
```

```
def nbrespairs(n):
    L=[2*k for k in range(n)]
    return L
```

```
def nbrespairs(n):
    L=[k for k in range(0,2*n) if x%2==0]
    return L
```

```
def nbrespairs(n):
    L=[k for k in range(0,2*n,2)]
    return L
```

`range(a,b,c)` : On va de a à $b-1$ avec un pas de c.

2e idée : utilisation d'une boucle while

2°)

```
def nbresimpairs(n):
    L=[2*k+1 for k in range(n)]
    return L
```

2

1°)
On peut utiliser une boucle for.

```
def pairsentre2bornes(a,b):
    L=[k for k in range(a,b+1) if k%2==0]
    return L
```

On peut utiliser une boucle while.

```
def pairsentre2bornes(a,b):
    L=[]
    k=a
    while k%2==0 and k<=b:
          L.append(k)
    return L
```

2°)

```
def impairsentre2bornes(a,b):
    L=[k for k in range(a,b+1) if k%2==1]
    return L
```

On peut aussi écrire la condition `k%2!=0`.

3

Il s'agit d'un cas particulier de l'exercice précédent.

```
def pairs(n) :
    L=[k for k in range(0,n+1,2)]
    return L
```

```
def impairs(n) :
    L=[k for k in range(1,n+1,2)]
    return L
```

4

Écrire une fonction `convert(n,b)` qui convertit un entier naturel n (donné par son écriture en base dix) en base b où b est un entier naturel supérieur ou égal à 2.

```
def convert(n,b):
    L=[]
    while n!=0:
          L.append(n%b)
          n=n//b
    return L
```

Il faut remettre les chiffres dans l'ordre.

```
def convert(n,b):
    L=[]
    x=[]
    while n!=0:
          L.append(n%b)
          n=n//b
    for i in range(1,len(L)):
          x.append(L[-i])
    return x
```

2 Algorithme avec boucle et test d'arrêt

Entrée :
Saisir n

Initialisation :
u prend la valeur n

Traitement :
Tantque $u \geqslant 11$ **Faire**
    u prend la valeur $u-11$
FinTantque

Sortie :
Afficher u

Recopier cet algorithme dans un cadre

Dans l'exercice, il n'est pas nécessaire de faire le programme sur calculatrice.

1°) On fait fonctionner l'algorithme « à la main » pour $n=35$, $n=50$, $n=55$.
On doit suivre l'algorithme pas à pas.
On fait tourner l'algorithme pas à pas.

On adopte la présentation donnée dans l'énoncé grâce à un tableau d'évolution des variables (méthode à retenir quand on doit faire tourner « à la main » un algorithme avec une boucle « Pour » ou « Tantque »).

On n'oublie pas de faire une phrase réponse à chaque fois pour donner la valeur de la variable u qui s'affiche en sortie.

Pour $n=35$

Étape	**0**	**1**	**2**	**3**	**4**
Condition $n \geqslant 11$		vraie	vraie	vraie	fausse
Valeur de n	35	24	13	2	

La valeur affichée en sortie est donc 2.

Pour $n=50$

Étape	**0**	**1**	**2**	**3**	**4**	**5**
Condition $n \geqslant 11$		vraie	vraie	vraie	vraie	fausse
Valeur de n	50	39	28	17	6	

La valeur affichée en sortie est donc 6.

Pour $n=55$

Étape	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Condition $n \geqslant 11$		vraie	vraie	Vraie	vraie	vraie	fausse
Valeur de n	55	44	33	22	11	0	

La valeur affichée en sortie est donc 0.

Commentaires :

1. La première case est barrée car u n'existe pas quand l'algorithme commence.
La condition $u \geqslant 11$ n'a donc pas lieu d'être.

2. Une fois que la condition est fausse, l'algorithme s'arrête. La variable u n'a plus de valeur.
On n'écrit donc aucune valeur dans la case.

3. Le tableau se lit de bas en haut.

Étape	**0**	**1**	**2**	**3**	**4**	**5**
Condition $n \geqslant 11$		vraie	vraie	vraie	vraie	faux
Valeur de n						

4. On a décidé de dénommer les étapes « étape 0 », « étape 1 », « étape 2 ».
Peut-être serait-il préférable d'appeler les rubriques « avant le premier passage dans la boucle », « après le premier passage dans la boucle » (et donc « avant le deuxième passage dans la boucle »), « après le deuxième passage dans la boucle », …, « après le dernier passage dans la boucle »…

Version plus courte de rédaction :

• Pour $n = 35$, la valeur finale de u affichée en sortie est 2.

• Pour $n = 50$, la valeur finale de u affichée en sortie est 6.

• Pour $n = 55$, la valeur finale de u affichée en sortie est 0.

Version plus courte mais à éviter (prise sur la feuille de Timothée Savouré, élève de TS1, le jeudi 1er octobre 2015) :

• Pour $n = 35$, $u = 2$.

• Pour $n = 50$, $u = 6$.

• Pour $n = 55$, $u = 0$.

Autre présentation :

Étape	0	1	2	3	4	5	6	7	8	9	10
Valeur de u	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$	$\frac{1}{256}$	$\frac{1}{512}$	$\frac{1}{1024}$
Valeur de n	0	1	2	3	4	5	6	7	8	9	10
Condition $u \geqslant 0,001$	V	V	V	V	V	V	V	V	V	V	F

2°) **Pour un entier naturel n quelconque, le nombre de sortie est égal au reste de la division euclidienne de n par 11.**
Il faudrait normalement démontrer deux points :
- la terminaison de l'algorithme (c'est-à-dire que l'algorithme s'arrête) ;
- la validation de l'algorithme (preuve de l'algorithme) c'est-à-dire qu'il donne bien le reste de la division euclidienne de n par 11.

On peut aussi parler du coût de cet algorithme (comparaison de la performance de cet algorithme par rapport à l'algorithme donné dans le cours). Cet algorithme nécessite beaucoup plus d'opérations que l'algorithme qui consiste à faire une division (et ce, d'autant plus que le nombre est plus élevé).

On peut réaliser le programme sur calculatrice (mais l'intérêt est assez limité car il permet seulement d'obtenir le reste de la division euclidienne par 11 donc programme à effacer tout de suite !).

À mettre en forme :

L'algorithme affiche en sortie un nombre $x \in [0\,;10]$.
$x = n - 11k$ où n est l'entier saisi en entrée et k le nombre de boucles effectuées par l'algorithme.
On a $n = 11k + x$.
Donc x est le reste de la division euclidienne de n par 11.

Programme sur calculatrice TI :

```
: Prompt N
: N → U
: While U ⩾ 11
: U – 11 → U
: End
: Disp U
```

Remarque :

On n'est pas obligé de mettre deux variables dans ce programme (ni dans l'algorithme).
Cependant il est plus commode d'utiliser 2 variables plutôt qu'une pour la lisibilité du programme.

Commentaires :

• Les opérations sont plus simples mais il y a plus de calculs (le coût de l'algorithme est plus élevé).

• Il est possible d'obtenir le quotient en introduisant une variable q initialisée à 0 puis en insérant l'instruction « q prend la valeur $q+1$ ».

• **Que se passe-t-il si on donne à n la valeur 9 en entrée ?**

La condition $n \geqslant 11$ n'est pas vérifiée.
On sort tout de suite de la boucle.
La valeur affichée en sortie est 9.

2°) La fonction `ex(n,a)` renvoie le reste de la division euclidienne de n par a.

3°)
Programme Python :

```
def ex(n,a):
    while n>=a:
        n=n-a
    return n
```

6 Années bissextiles

1°)

année	bissextile ?
2020	oui
20134	non
2100	non
2000	oui
2016	oui
1900	non

2°)

Algorithme fondamental :

Si ($4 \mid a$ et $100 \nmid a$) ou $400 \mid a$
  Alors afficher "année bissextile"
  Sinon afficher "année non bissextile"
FinSi

Fonction Python :

```
def bissext(a):
    if (a%4==0 and a%100!=0) or a%400==0:
       return True
    else:
       return False
```

On peut proposer une version booléenne.

```
def bissext(a):
    cas1=a%4 == 0 and a%100 != 0
    cas2=a%400 == 0
    return cas1 or cas2
```

On peut aussi utiliser la structure Python « if … elif … else » (version extraite d'un livre de NSI notée le 11-12-2020).

```
def bissext(a):
    if a%4==0 and a%100!=0:
       bissext=True
    elif a%400=0:
       bissext=True
    else:
       bissext=False
    return bissext
```

On peut éventuellement utiliser la fonction test_divis(x,a) du cours.

```
def bissext(a):
    if a%4==0 and a%100!=0:
       print(a,« est une année bissextile »)
    elif a%400=0:
       print(a,« est une année bissextile »)
    else:
       print(a,« n'est pas une année bissextile »)
    return bissext
```

```
def bissext(a):
    if a%4==0 and a%100!=0:
       print(f « {a} est une année bissextile »)
    elif a%400=0:
       print(f « {a} est une année bissextile »)
    else:
       print(f « {a} n'est pas une année bissextile »)
    return bissext
```

possibilité d'utiliser un or plutôt que le elif.

3°)

On utilise la fonction `bissext(a)` créée à la question 2°).

```
def liste_bissext(a,b):
    L=[]
    for i in range(a,b+1):
        if bissext(i)==True:
           L.append(i)
    return L
```

`L=[]` :
[ ] est une liste vide qu'on va remplir avec l'algorithme.
append en anglais signifie « ajouter »

```
def liste_bissext(a,b):
    L=[x for x in range(a,b+1) if bissext(x)==True]
    return L
```

La liste des années bissextiles entre 2000 et 2100 est :

[2000, 2004, 2008, 2012, 2016, 2020, 2024, 2028, 2032, 2036, 2040, 2044, 2048, 2052, 2056, 2060, 2064, 2068, 2072, 2076, 2080, 2084, 2088, 2092, 2096]

4

Nombre de diviseurs positifs d'un entier naturel non nul

$\forall n \in \mathbb{N}^*$ $d(n)$ = nombre de diviseurs positifs de n

On peut écrire $d(n) = \text{card}\, D^+(n)$ (ou encore $d(n) = \text{card}\{k \in \mathbb{N} / k \mid n\}$).

$d(n)$ est le cardinal de $D^+(n)$. On rappelle que le cardinal d'un ensemble fini E, noté card E, désigne le nombre de ses éléments. On peut écrire $d(n) = \text{card}\, D^+(n)$.

Si p est un nombre premier, $d(p) =$?

Caractériser les entiers naturels n tels que $d(n) = n+1$.

1°) **Tableau de valeurs**

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$d(n)$	1	2	2	3	2	4	2	4	3	4	2	6	2	4	4	5

Commentaires sur ce tableau :

- On peut compléter ce tableau « à la main » ou en utilisant le programme qui est réalisé dans la question 2°).
- On observe que certains entiers ont exactement 2 diviseurs positifs : ce sont les nombres premiers.
- On observe que certains entiers ont exactement 3 diviseurs positifs : ce sont les carrés des nombres premiers.

Question supplémentaire possible :

Le 3-10-2014

Caractériser les entiers naturels qui admettent exactement 4 diviseurs positifs ou nuls.

Il y a les nombres qui s'écrivent comme produit de deux nombres premiers distincts ou comme cube d'un nombre premier.

2°)

Algorithme fondamental :

```
d ← 0
Pour i allant de 1 à n Faire
    Si i | n
        Alors d ← d +1
    FinSi
FinPour
```

Fonction Python

Dans le programme de droite, on utilise la fonction `len` qui donne la longueur de la liste, c'est-à-dire le nombre d'éléments (length en anglais). Exemple : `len('hello')=5`.

```
def d(n):
    d=0
    for i in range(1,n+1):
        if n%i==0:
            d=d+1
    return d
```

```
def d(n):
    L=[i for i in range(1,n+1) if n%i==0]
    d=len(L)
    return d
```

Il s'agit d'un algorithme de comptage ou de dénombrement.

L'algorithme utilise 3 variables : n, d, i (la variable i est une variable interne ou locale ; elle n'intervient ni dans l'entrée ni dans la sortie).

b) Réaliser le programme correspondant sur calculatrice.

c)

- On lance le programme pour $n = 1000$.

On obtient en sortie le nombre 16.
Donc $d(1000) = 16$. Donc 1000 a 16 diviseurs positifs.

- Le plus petit entier naturel qui admet exactement 15 diviseurs positifs est 144.

- On peut gagner du temps en observant que 15 est impair donc que l'entier cherché est un carré parfait.

- De manière générale, nous démontrerons plus tard dans un exercice sur les nombres premiers que les entiers naturels qui admettent exactement 15 diviseurs positifs sont les entiers naturels de la forme p^{14} où p est un nombre premier ou de la forme p^2q^4 où p et q sont deux nombres premiers distincts.

On ne peut pas trouver une expression explicite de $d(n)$ en fonction de n d'où l'intérêt d'avoir un algorithme.

On verra plus tard dans le chapitre sur les nombres premiers comment calculer le nombre de diviseurs positifs d'un entier naturel non nul à partir de sa décomposition en facteurs premiers.

3°)

`int(n%i==0)` prend la valeur 1 si la condition à l'intérieur de la parenthèse est vérifiée et 0 sinon.

```
def d(n):
    L=[int(n%i==0) for i in range(1,n+1)]
    d=sum(L)
    return d
```

On peut utiliser la fonction $Y1=\sum_{K=1}^{K=X}(\text{remainder}(X,K)=0)$.

On obtient un tableau de valeurs.

Question enlevée le 2-12-2023 :

3°) **Utilisation de la calculatrice TI Premium CE**
Retrouver les résultats de la question précédente en rentrant dans la calculatrice la fonction

$Y1=\sum_{K=1}^{X}(\text{reste}(X,K)=0)$ ou $Y1=\sum_{K=1}^{X}(\text{remainder}(X,K)=0)$.

L'expression $(\text{reste}(X,K)=0)$ est une expression booléenne qui prend la valeur 1 si le reste de la division euclidienne de X par K vaut 0 (autrement dit, si X est divisible par K) et 0 dans le cas contraire.

On tape le X avec la touche X,T,θ,n .

8 Somme des diviseurs positifs d'un entier naturel non nul

$\forall n\in\mathbb{N}^*\quad S(n)=$ somme des diviseurs positifs de n

On peut écrire $S(n)=\sum_{\substack{k\in\mathbb{N}\\ k\mid n}}k$.

Algorithme fondamental :

S ← 0
Pour *i* allant de 1 à *n* **Faire**
  Si *i* | *n*
    Alors S ← S+*i*
  FinSi
FinPour

Fonction Python :

```
def S(n):
    s=0
    for i in range(1,n+1):
        if n%i==0:
            s=s+i
    return s
```

```
def S(n):
    L=[i for i in range(1,n+1) if n%i==0]
    s=sum(L)
    return s
```

```
def S(n):
    L=[i for i in range(1,n+1) if n%i==0]
    return sum(L)
```

```
def S(n):
    L=[i*int(n%i==0) for i in range(1,n+1)]
    s=sum(L)
    return s
```

Le mercredi 4-1-2023

Pour $n=6$ (`range(1,7)`)

	i	x
Initial		0
Après la 1ère boucle	1	1
Après la 2e boucle	2	$1+2=3$
Après la 3e boucle	3	$3+3=6$
Après la 4e boucle	4	6
Après la 5e boucle	5	6
Après la 6e boucle	6	$6+6=12$

8bis Nombres abondants, déficients, parfaits

```
def S(n):
    s=0
    for i in range (1,n):
        if n%i==0:
            s=s+i
    return s

def nature (n):
    if S(n)>n:
        print("Abondant")
    if S(n)==n:
        print("Parfait")
    if S(n)<n:
       print("Déficient")
```

Chiffrement de Hill

Étape 1 :

ST correspond à $(x_1 ; x_2) = (18 ; 19)$.

Étape 2 :

• $11x_1 + 3x_2 = 11\times18 + 3\times19 = 198 + 57 = 255$
y_1 est alors le reste de la division euclidienne de 255 par 26.
Comme $255 = 9\times26 + 21$ et que $0 \leqslant 21 < 26$, on en déduit que $y_1 = 21$.
• $7x_1 + 4x_2 = 7\times18 + 4\times19 = 126 + 76 = 202$. Comme $202 = 7\times26 + 20$ et que $0 \leqslant 20 < 26$, on en déduit que $y_2 = 20$.

Étape 3 :

Le couple $(21 ; 20)$ correspond au mot VU et donc le mot ST se code en VU.

Il est possible de créer un petit programme de codage sur calculatrice.

Le 28 octobre 2016

Chiffrement de Hill

La **fonction de codage** est la fonction $f : (x ; y) \mapsto (x' ; y')$ avec
x' : reste de la division euclidienne de $11x + 3y$ par 26 ;
y' : reste de la division euclidienne de $7x + 4y$ par 26.

```
def chiffrement(letter):
    alphabet
=["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T"
,"U","V","W","X","Y","Z"," "]
    return alphabet.index(letter)

def dechiffrement(number):
    alphabet
=["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T"
,"U","V","W","X","Y","Z"," "]
    return alphabet[number]

def hill(letters):
    x=(chiffrement(letters[0]),chiffrement(letters[1]))
    y = ((11*x[0]+3*x[1])%26,(7*x[0]+4*x[1])%26)
       return dechiffrement(y[0])+dechiffrement(y[1])
```

Auguste Charpentier le 1-12-2020 (élève de terminale maths experts)

```
def hill(text):
    result = ""
    for i in range(0, len(text), 2):
        a = text[i]
        b = text[i+1]
        a = letters.index(a)
        b = letters.index(b)
        a = (11*a + 3*b) % 26
        b = (7*a + 4*b) % 26
        result = result + letters[a] + letters[b]
  return result

print(hill("MATH"))
```

Auguste Charpentier le 1-12-2020 (élève de terminale maths experts)

```
def hill(text):
    result = ""
    for i in range(0, len(text), 2):
        a = text[i]
        b = text[i+1]
        a = letters.index(a)
        b = letters.index(b)
        a = (11*a + 3*b) % 26
        b = (7*a + 4*b) % 26
        result = result + letters[a] + letters[b]
  return result

print(hill("MATH"))
```

Commentaires pour le programme d'Auguste Charpentier notés le 14-12-2020

```
1.  letters = "ABCDEFHIJKLMNOPQRSTUVWXYZ"
2.
3.
4.  def hill(text):
5.      result = ""  # notre futur texte codé
6.
7.      # on rentre dans une itération avec un pas de 2 car on
8.      # travaille avec des paquets de 2 lettres dans le texte.
9.      # len(text) renvoie la longueur du texte, donc 4 avec "MATH"
10.     for i in range(0, len(text), 2):
11.
12.         # on obtient les deux lettres suivantes dans notre texte
13.         a = text[i]
14.         b = text[i + 1]
15.
16.         # première étape : on obtient le nombre correspondant à la
  lettre
17.         a = letters.index(a)
18.         b = letters.index(b)
19.
20.         # deuxième étape : on effectue la division euclidienne
21.         a = (11 * a + 3 * b) % 26
22.         b = (7 * a + 4 * b) % 26
23.
24.         # troisième étape : on réutilise nos lettres pour construire
  le mot, qu'on ajoute à result
25.         result = result + letters[a] + letters[b]
26.
27.     return result
28.
29.
30. print(hill("MATH"))
31. >>> SQJC
```

Dans ce programme, on va progressivement coder notre texte par paquets de 2.

On rentre dans une boucle for allant de 0 à n, la longueur de notre texte, avec un pas de 2.
Si on a un texte de 6 caractères, la boucle sera exécutée 3 fois, avec i valant 0 puis 2 puis 4.

Ensuite, on va obtenir `a` et `b`, les deux lettres avec lesquelles on va travailler.
En faisant `"MATH"[2]`, *on va obtenir* `"T"` *(le compte commençant à 0)*
On va donc aller prendre `text[i]` pour la première lettre `a` et `text[i + 1]` pour la deuxième lettre `b`.

Première étape
On cherche à traduire `a` et `b` en fonction de la position de la lettre dans l'alphabet.

On a `letters` qui est une chaîne de caractères pour tout l'alphabet.
La fonction `index` nous permet de nous donner la position de la première occurrence dans le texte.
`letters.index` donne donc la position d'une lettre de 0 à 26.

On a donc `a` et `b` qui sont égaux à x_1 et x_2.

Deuxième étape
On cherche à traduire `a` et `b` selon les divisions euclidiennes, traduites ainsi en Python :

```
1.  a = (11 * a + 3 * b) % 26
2.  b = (7 * a + 4 * b) % 26
```

L'opérateur `%` correspond au modulo, le reste de la division de a par b.

On a donc `a` et `b` qui sont égaux à y_1 et y_2.

Troisième étape
On cherche à traduire `a` et `b` pour qu'ils soient égaux à une nouvelle lettre dans l'alphabet.

On effectue la même opération `letters[x]` tel que x est la position de la lettre de 0 à 26 dans l'alphabet.
On additionne nos deux lettres ensemble, puis on les ajoute à `result`.

Enfin, on renvoie `result`.

9

```
def nbc(n):
    x = 1
    while n>= 10:
          n = n // 10
          x = x + 1
    return x
```

1°) Recopier cette fonction.
Faire tourner cette fonction à la main pour plusieurs valeurs de *n*.

$n = 407$

Un méthode qui permet de bien comprendre est d'utiliser un tableau d'évolution des variables *n* et *x*.

Étape	0	1	2	3
Condition $n \leqslant 10$		V	V	F
Valeur de *n*	407	40	4	
Valeur de *x*	1	2	3	

Pour $n = 407$, la valeur de *x* en sortie est 3.

La fonction `nbc(n)` renvoie le nombre de chiffres de l'écriture en base dix de *n*.

Que se passe-t-il si $n = 0$, 1, 2, …, 9 ?

Si on met un nombre non entier, par exemple π, affichage : 1 (nombre de chiffres de la partie entière).

Un raccourci : `print(len(str(n)))`

On sait que le nombre de chiffres de l'écriture en base dix d'un entier naturel $n \geqslant 1$ est égal à $\mathrm{E}(\log n)+1$.
Ce n'est pas ce qu'on utilise ici.

2°) Programmer cette fonction sur calculatrice.

Pour aller plus loin : fonction qui affiche le nombre de « chiffres » d'un entier naturel *n* dans une base *b* quelconque (*b* entier naturel supérieur ou égal à 2)

```
def nbc(n,b):
    x = 1
    while n>= b:
          n = n // b
          x = x + 1
    return x
```

On peut obtenir ainsi le nombre de chiffres dans n'importe quelle base (base deux, base huit, base neuf, base hexadécimale, base soixante…).

`nbc(n,b)` : 2 arguments séparés par une virgule.
n est un entier naturel
b désigne la base.
On cherche le nombre de chiffres de l'écriture de *n* en base *b*.

Hugo Deschamps le 18-1-2022 :

```
n=int(input(« votre nombre : »))
b=int(input(« Votre base : »))
print(« Le nombre », n, « est constitué de », nbc(n,b), « chiffres en base », b)
```

10 Numéro INSEE et clé de contrôle

C = 97 – reste de la division euclidienne de N par 97

1°)

```
def cle(m):
    c=97-m%97
    return c
```

```
def cle(m):
    c=97-m%97
    if c<10:
       return (“0”+str(c))
    else:
       return c
```

2°)

```
def test(n):
    cle=x%100
    char=x//100
    if cle(char)==cle:
       return True
    else:
       return False
```

11

$n = \overline{ab}$

1°)
a est le quotient de la division euclidienne de *n* par 10.
b est le reste de la division euclidienne de *n* par 10.

2°)

```
def reverse(n):
    a=n//10
    b=n%10
    m=10*b+a
    return m
```

13

```
def retourne(x) :
    y = 0
    while x != 0:
          y = (y * 10) + (x % 10)
          x = x // 10
    return y
```

1°)

On fait fonctionner la fonction pour 347.

$y = 0$

1ère boucle :
$y = 7$
$x = 34$

2e boucle :
$y = 70 + 4 = 74$
$x = 3$

3e boucle :

$y = 740 + 3 = 743$
$x = 0$

On extrait les chiffres de l'entier naturel de départ un à un.
Puis on reconstitue le retourné.

python.developpez.com

14

```
def al(n) :
    u=100
    for _ in range(n):
        u=(15091*u)%64007
    return u
```

On peut aussi écrire `u=15091*u%64007`. Il cependant préférable de les mettre.

La fonction fournit un entier naturel pseudo-aléatoire inférieur ou égal à 64006.
u_0 s'appelle la « graine ».

$u_{30} = 42782$
$u_{300} = 35181$
$u_{3000} = 36849$

14 Générateur d'entiers pseudo-aléatoires

On considère la suite (u_n) d'entiers naturels définie sur $\mathbb{N}$ par son premier terme $u_0 = 100$ et u_n = reste de la division euclidienne de $15091 \times u_{n-1}$ par 64007 pour tout entier naturel $n \geqslant 1$.
Programmer le calcul des termes de la suite (u_n) en Python.
Que valent u_{30} ? u_{300} ? u_{3000} ?

$u_0 = 100$
$u_{n+1} = \text{rem}(15091.u_n, 64807)$

Il s'agit d'un exemple de générateur congruentiel linéaire.

15

Somme des chiffres de l'écriture en base dix d'un entier naturel

```
def SommeChiffres(n):
    s = 0
    while n!=0:
          x = n % 10
          s = s + x
          n = n//10
    return s
```

Autre proposition (Vicente Seixas le mardi 18-1-2022) :

```
def SommeChiffres(n):
    u=0
    j=str(n)
    for i in range(len(j)):
        u=int(j[i])+u
    return u
```

16

Il y a deux versions possibles.

Version 1 :

```
def recherche_cubesv1() :
    L=[]
    for x in range(100,1000) :
        a = x%10
        j = x//10
        b = j%10
        c = j // 10
          if a**3 + b**3 + c**3 == x :
          L.append(x)
     return L
```

Version 2 :

```
def recherche_cubesV2() :
    L=[]
    for a in range(10) :
        for b in range(10) :
            for c in range(10) :
                if a**3+b**3+c**3 == a*100 + b*10 + c and len(str(a*100 +
 b*10 + c)) == 3 :
                L.append(== a*100 + b*10 + c)
    return L
```

`L = [153,370,371,407]`

17

Fonction Python qui affiche le premier chiffre de l'écriture en base dix d'un entier naturel

```
def firstdigit(n):
    while (n >= 10):
        n=n//10
    return n
```

On peut faire le lien avec la loi de Benford.

18

Écrire une fonction Python d'en tête `def suppression(L):` qui prend pour argument une liste L et qui renvoie la liste constituée des éléments non divisibles par 5.

```
def suppression(L):
    for x in L:
        if x%5==0:
            L.remove(x)
    return L
```

19

Écrire une fonction Python d'en-tête `racines_empilées(L):` qui prend en argument une liste $L=[a_0, a_1, \ldots, a_n]$ de réels positifs ou nuls et qui renvoie la valeur de $\sqrt{a_n+\sqrt{a_{n-1}+\ldots+\sqrt{a_0}}}$.
Tester la fonction avec une liste composée que de 0 ou que de 1.

2 propositions.

Dans les deux cas, on utilise une boucle for.

La fonction sqrt doit être préalablement importée du module math.

1ère proposition :

```
def racines_empilées(L):
    r=0
    for x in L:
        r=sqrt(x+r)
    return r
```

2e proposition :

```
def racines_empilées(L):
    r=0
    for k in range(len(L)):
        r=sqrt(r+L[k])
    return r
```

On effectue les tests demandés.

- Pour une liste composée uniquement de 0, on obtient 0 pour résultat (c'est évident).
- Pour une liste composée uniquement de 1, on obtient un résultat qui se rapproche du nombre d'or $\varphi=\frac{1+\sqrt{5}}{2}$ (nombre irrationnel) lorsque la longueur de la liste augmente. On a en effet la formule :

$$\varphi=\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{\ldots}}}}\ .$$

20

Écrire une fonction Python d'en-tête `fractions_empilées(L):` qui prend en argument une liste $L=[a_0, a_1, \ldots, a_n]$ de réels positifs ou nuls et qui renvoie la valeur de $\cfrac{1}{a_n+\cfrac{1}{\ldots\ldots}}$.

$$a_1+\cfrac{1}{a_0}$$

Le 11 mars 2022

$$\frac{1}{a_0} \qquad \frac{1}{a_1+\frac{1}{a_0}} \qquad \frac{1}{a_2+\cfrac{1}{a_1+\frac{1}{a_0}}}$$

Tester la fonction avec une liste composée que de 1.

2 propositions.

Dans les deux cas, on utilise une boucle for.

1ère proposition :

```
def fractions_empilées(L):
    s=0
    for x in L:
        s=1/(x+s)
    return s
```

2e proposition :

```
def fractions_empilées(L):
    s=0
    for k in range(len(L)):
        s=1/(s+L[k])
    return s
```

On effectue les tests demandés.

Le 11-3-2023

Fractions continues

$$a_0+\cfrac{1}{a_1+\cfrac{1}{a_2+\cfrac{1}{\ddots}}}$$

$$a_{n-1}+\frac{1}{a_n}$$

math.unice.fr/L1Arithmcours2.pdf

$\frac{1}{1}\ \frac{3}{2}\ \frac{8}{5}\ \frac{21}{13}$

$\pi=[3,7,15,1,292,1,1,1,2,...]$

21 Écriture en base deux d'un entier naturel

```
def maxpuis(n) :
    p=0
    while 2**p<=n :
          p=p+1
    return p-1

def decomp(n) :
    L=[]
    while n !=0 :
         L.append(2)
         L.append(maxpuis(n))
         n=n-2**(maxpuis(n))
    return L
```

Il s'agit d'un algorithme glouton.

22

Écrire une fonction Python qui prend en argument un entier naturel *n* supérieur ou égal à 1 et qui renvoie la fraction d'entiers naturels non nuls …
Affichage de toutes les fractions :

```
from math import abs, pi

def approx_fract(n):
    e=10
    for a in range (1,n) :
        for b in range (1,n):
            if abs(pi-a/b)<e:
                e=abs(pi-a/b)
                print(a,"/",b)
```

Affichage de la dernière fraction :

```
from math import abs, pi

def approx_fract(n):
    e=10
    for a in range (1,n) :
        for b in range (1,n):
            if abs(pi-a/b)<e:
                e=abs(pi-a/b)
                n=a
                d=b
                print(n,"/",d)
```

23

Écrire une fonction Python d'en-tête `def nbr_pairs(L):` qui prend pour argument une liste L d'entiers relatifs et qui renvoie le nombre de nombre pairs de la liste.

```
def nbr_pairs(L):
    c=0
    for x in L:
        if x%2==0:
        c=c+1
    return c
```

On utilise une variable *c* qui est un compteur.

24

Écrire une fonction Python d'en-tête `def nbr_posstrict(L):` qui prend pour argument une liste L de réels et qui renvoie le nombre de réels strictement positifs.

```
def nbr_posstrict(L):
    c=0
    for x in L:
        if x>0:
        c=c+1
    return c
```