TS spé

Algorithmes liés à la divisibilité et à la division euclidienne

Le mardi 4-1-2023

Plan du chapitre

Compétences :

- Connaître toute la partie **I** (définition précise).
- Connaître les commandes de la calculatrice permettant de faire une division euclidienne sur calculatrice.
- Savoir écrire sans difficulté le programme donnant les diviseurs positifs d'un entier naturel non nul de la **partie C**, **I. 2°)**.


```
L=[1]
for k in range(2,n+1):
    if n%k==0:
       L.append(k)
return L
```

Pour commencer : Quelques rappels de base sur le langage Python

1. Autour du signe =

- Le signe « = » correspond à l'affection d'une valeur à une variable.

Exemple : L'instruction `a=3` signifie « *a* (la variable) prend la valeur 3 ». Le signe = est orienté. On ne peut pas écrire `3=a`.

-Le signe « == » (deux signes d'égalités) est un test d'égalité (utilisé dans les boucles if et while).

Le signe « != » correspond au $\neq$ mathématique.

Les opérateurs de comparaison standards sont écrits :
< (inférieur), > (supérieur), == (égal), <= (inférieur ou égal), >= (supérieur ou égal) et != (non égal, différent).

Attention, pour <= et >=, le = toujours à droite.

Connecteurs :

ou : « or » et : « and »

Le 9-1-2022

Cadre de la logique qu'on utilise en Python.

Principe de négation

Test dans une boucle while :

On se réfère à la négation d'une proposition.

Exemples (fondamentaux) :

assertion	négation
$a = b$	$a \neq b$
$a < b$	$a \geqslant b$
$a \leqslant b$	$a > b$

2. Les structures :

Il existe 2 structures très utilisées :

Structures conditionnelles

- Le « if ». Il introduit une condition. On peut l'introduire sous la forme d'une simple condition avec « if … » ou mettre un sinon en écrivant :

```
if … :
else … :
```

On pourra aussi noter plusieurs conditions avec la structure « `if… : elif… : else… :` » (elif : contraction de else et if). Il a la structure suivante :

```
if … :
elif … :
else … :
```

Structures de boucles

La deuxième structure la plus commune est la boucle de répétition. On a 2 variantes de ces boucles. La boucle « for » (pour) qui est une répétition faite un nombre fixe au sein du programme et la boucle « while » (tant que) qui s'achève lorsqu'une condition est remplie.

La boucle for s'écrit « `for i in range(1,n+1):` » si l'on veut le répéter *n* fois, car Python considère que *n* consiste à faire $n-1$ répétitions. On peut aussi la noter « `for i in range(n):` » (qui équivaut à `range(0,n)`) où l'on rentre le nombre de répétition que l'on veut faire.

On peut aussi noter « `for i in range(a,b,c):` ».
`a` : début
`b` : « fin » (plutôt $b-1$)
`c` : « pas »
On n'oubliera pas de faire attention à écrire le « : » à la fin de chaque instruction.

La boucle « while »
On la note dans le programme de cette façon : « while [condition] : ».

3. Listes

Notation :

Une liste en Python se note entre crochets. Les éléments sont séparés par des virgules.
Exemple : `L=[2,3,5]` (liste des 3 premiers nombres premiers).

Cas particulier :

La liste vide, notée avec 2 crochets sans éléments (un crochet ouvrant et un crochet fermant) : `L=[]` .

La notion de liste intervient quand on veut créer (remplir) une liste à l'aide d'une instruction conditionnelle ou d'une boucle (`for` ou `while`). C'est le cas de l'un des programmes qui donne la liste des diviseurs positifs d'un entier.

Numérotation des éléments d'une liste en Python

Les éléments d'une liste sont numérotés à partir de 0.

Écriture d'une liste par compréhension

Exemple : `L=[k**2 for k in range(0,4)]` donne la liste des carrés des entiers naturels de 0 à 3.

Fonction append()

Exemple : `L.append(5)`

Cette commande rajoute un élément en bout de liste.

Cette fonction sert à créer des listes dans des boucles par ajout d'éléments (voir l'un des programmes qui donne la liste des diviseurs positifs d'un entier).

Fonction remove()

Exemple : `L.remove(5)`

Cette commande permet d'enlever un élément dans une liste (un seul à la fois, le premier qu'elle rencontre dans la liste).

Opération + :

Permet de rajouter plusieurs éléments à une liste. C'est une opération de concaténation, c'est à dire que l'on ne peut pas ajouter de listes entre elles.

Exemple : `L=L+[4,5]`

$L = L + [4,5]$

Les éléments sont concaténés à la fin.

Fonctions sorted( ) et sort() :

On peut trier des listes de nombres dans l'ordre croissant de 2 façons :

- La fonction native `sorted()` qui trie les éléments d'une liste par ordre croissant.

Exemple :
```
L=[7,2,9]
sorted(L)
L=[2,7,9]
```


- La commande `list.sort()`. C'est une commande native.

Exemple :
```
L=[7,6,9]
list.sort(L)
[6,7,9]
```

Fonction len()

length : longueur en anglais
*
Elle donne le nombre d'éléments de la liste que l'on appelle longueur de la liste.

Fonction sum() :

sum(L) : somme de tous les éléments de la liste L.
Il n'y a pas de fonction pour le produit (existe dans la bibliothèque numpy).

Fonction min : `min(L)`

Fonction max : `max(L)`

Le 26 décembre 2022

Produit des éléments d'une liste

① tresfacile.net https://www.tresfacile.net/solution-exercice-27-fonction-python-qui-calcul-la-somme-et-le-produit-des-elements-dune-liste/

Exercice 27 : fonction Python qui calcule la somme et le produit des éléments d'une liste
Écrire un programme en Python sous forme de fonction qui calcul la somme des éléments d'une liste de nombres. Et un autre qui permet de multiplier tous les éléments d'une liste de nombres.

Solution

5. Fonctions

Intérêt des fonctions : appeler une fonction à l'intérieur d'une autre

Voir cours de Frédéric Peurière

Deux programmes fondamentaux

Fonction qui calcule la somme des éléments d'une liste (sans utiliser la fonction sum)

```
# fonction qui calcule la somme des éléments d'une liste
def somme(L):
    s=0
    # création de la somme en parcourant les éléments de la liste
    for x in L:
        s=s+x
    return s
```

```
somme([2,5,3])
10
```

Autre proposition à savoir écrire également :

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

Fonction qui calcule le produit des éléments d'une liste

```
# fonction qui calcule le produit des éléments d'une liste
def produit(L):
    p=1
    # création du produit en parcourant les éléments de la liste
    for x in L:
        p=p*x
    return p
```

```
produit([2,5,3])
30
```

Autre proposition à savoir écrire également :

```
def produit(L):
    p=1
    for k in range(len(L)):
        p=p*L[k]
    return p
```

On peut aussi noter la fonction produit et noter la variable p au lieu de m (plus cohérent par rapport à la somme).

② avec Numpy
np.prod(liste) voir DS1 24 septembre 2022 Frédéric Gaunard

Autres ressources :
③ • cpge.frama.io Numpy page bien

④ • Paris-Diderot cours Python : Paris tirer « Plus sur les listes » fonction remove

Le 26 décembre 2022

Enlever un élément dans une liste

2 programmes fondamentaux :
Puissances successives d'un réel
Factorielle d'un entier naturel

Le 29 décembre 2022

La bibliothèque mathplotlib

```
import matplotlib.pyplot as plt

X = [0,1,2,3]
Y = [3,0,5,1]
plt.plot(X,Y)
plt.title("essai")
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

Les points sont reliés entre eux.

Fonctions arithmétiques en Python voir plus loin

Le 1-12-2020

Nous prenons le chapitre sur algorithmes liés à la divisibilité et division euclidienne.
Le chapitre sera refait en classe.

3 fonctions Python importantes :

• x%y donne le reste de la division euclidienne de x par y pour x entier relatif et y entier naturel non nul.

• x//y donne le quotient de la division euclidienne de x par y pour x entier relatif et y entier naturel non nul.

• divmod(x,y) donne le couple (quotient, reste) pour x entier relatif et y entier naturel quelconque (commande qui équivaut à (x//y,x%y)).

• Une autre fonction à importer de la bibliothèque math :
floor(x) donne la partie entière de x pour x réel.

Utilisation : test de divisibilité

La fonction d'en-tête `def divisible(x,n) :` écrite dans le cadre ci-dessous prend deux arguments : x entier relatif et n entier naturel non nul.
Elle permet de tester si x est divisible par n.

```
def divisible(x,n):
    if x%n==0 :
       return True
    else:
       return False
```

Fonction booléenne : fonction Python qui renvoie True si x est divisible par a et False sinon.

```
def test_divisibilite(x,a):
    if x%a==0:
       return True
    else:
       return False
```


Le 2-12-2020

Programme fondamental :

On va écrire une fonction Python qui prend pour argument un entier naturel *n* non nul et qui renvoie la liste des diviseurs positifs (on fera plus loin une autre fonction qui inclura les diviseurs négatifs) de *n*.

$\boldsymbol{D}^{+}(n)=\{k\in\mathbb{N}\ /\ k\,|\,n\}$ (écriture de l'ensemble en compréhension)
On sait que $\boldsymbol{D}^{+}(n)\subset[\![1;n]\!]$.
On teste tous les entiers naturels de 1 à *n*.

Structure fondamentale en langage naturel :

Pour *k* allant de 1 à *n* **Faire**
  Si *k* | *n*
    afficher *k*
  FinSi
FinPour

Fonction Python qui renvoie tous les diviseurs positifs d'un entier naturel *n* supérieur ou égal à 1 :

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

On remarquera qu'il n'y a pas de else.

Autre version (sans fonction) :

```
n=int(input('Entrer le nombre :'))
for i in range(1,n+1):
    if n%i==0:
       print(i)
```

Autre version avec une boucle while :

```
def divipos(n):
    i=1
    while i<=n and n%i==0:
          print(i)
          i=i+1
```

On ne peut pas échanger les instructions `print(i)` et `i=i+1`.

Fonction Python qui renvoie la liste tous les diviseurs positifs d'un entier naturel *n* supérieur ou égal à 1 :

On utilise les listes.
On commence par créer une liste vide.

```
def divipos(n):
    L=[]
    for k in range(1,n+1):
          if n%k==0:
             L.append(k)
    return L
```

La fonction append permet d'ajouter un élément à une liste et un seul.
Pour en ajouter deux ou plus, on utilise la concaténation avec un +.

Fonction divipos(n):
L est une liste vide
Pour *k* allant de 1 à *n* **Faire**
  Si *k* divise *n*
    Alors ajouter *k* à la liste L
  FinSi
FinPour
Renvoyer L

On notera bien que si *k* ne divise pas *n*, alors on ne fait rien (la liste L ne change pas).

Il faut impérativement savoir écrire ce programme.

Version plus courte :

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

```
def divipos(n):
    L=[]
    for k in range(1,n+1):
          if n%k==0:
             L.append(k)
    return L
```

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

Ensemble de tous les diviseurs, positifs et négatifs

Il y a deux options :

① première idée : On teste tous les entiers naturels de – *n* à *n*.

```
def divi(n):
    for i in range(-n,n+1):
          if n%i==0:
             print(i)
```

Ce programme Python ne fonctionne pas à cause du 0.

Deuxième idée : On teste tous les entiers naturels de – *n* à *n* sauf 0.
On teste tous les entiers naturels de – *n* à – 1 de 1 à *n*.
On aura donc deux boucles for.

② On teste juste tous les entiers naturels de 1 à *n* et on adjoint à chaque fois leurs opposés.

```
def divi(n):
    for i in range(1,n+1):
          if n%i==0:
             print(i)
             print(-i)
```

Fonction Python qui renvoie la liste de tous les diviseurs positifs et négatifs d'un entier naturel *n* supérieur ou égal à 1 :

```
def divis(n):
    L=[]
    for k in range(1,n+1):
        if n%k==0:
           L=L+[k,-k]
    return L
```

Le 7-12-2022

Amélioration : utilisation des diviseurs associés

Rappels sur les diviseurs associés

Définition [diviseurs associés]

Soit n un entier relatif.
On dit que deux entiers relatifs d et d' sont deux diviseurs associés de n lorsque $dd' = n$.

Exemple :

2 et 4 sont deux diviseurs associés de 8 car $2 \times 4 = 8$.

Soit n un entier relatif.

Si d est un diviseur de n, alors son diviseur associé est $d' = \frac{n}{d}$.

n est un entier non nul.

Dès qu'on a un diviseur k de n, alors $\frac{n}{k}$ est aussi un diviseur de n (on notera que k est forcément non nul).

On effectue un affichage par doublets.

On utilise le lemme suivant des diviseurs associés positifs d'un entier naturel n.

Lemme :

Soit n un entier naturel non nul.
Si l'on a deux diviseurs positifs associés de n, le plus petit est inférieur ou égal à $\sqrt{n}$, le plus grand est supérieur ou égal à $\sqrt{n}$.

Règle pratique :

Pour déterminer la liste des diviseurs positifs d'un entier naturel non nul n, on teste (ou il suffit de tester) la divisibilité de n par tous les entiers naturels inférieurs ou égaux à $\sqrt{n}$.

On s'arrête au plus grand entier naturel inférieur ou égal à $\sqrt{n}$.


```
from math import sqrt, floor

def divpos(n):
    L=[]
    for k in range(1,floor(sqrt(n))+1):
        if n%k==0:
           L=L+[k,n//k]
    return L
```

On peut ordonner la liste dans l'ordre croissant.

Rappel pour ordonner une liste en Python :

→ simple tri
Il y a deux façons sorted(list) et list.sort()

→ tri inversé : sorted(list, reverse=True)

```
from math import sqrt, floor

def divpos(n):
    L=[]
    for k in range(1,floor(sqrt(n))+1):
        if n%k==0:
           L=L+[k,n//k]
    L.sort()
    return L
```

Il y a un souci dans les doublets qui apparaissent.

Le 14 décembre 2022

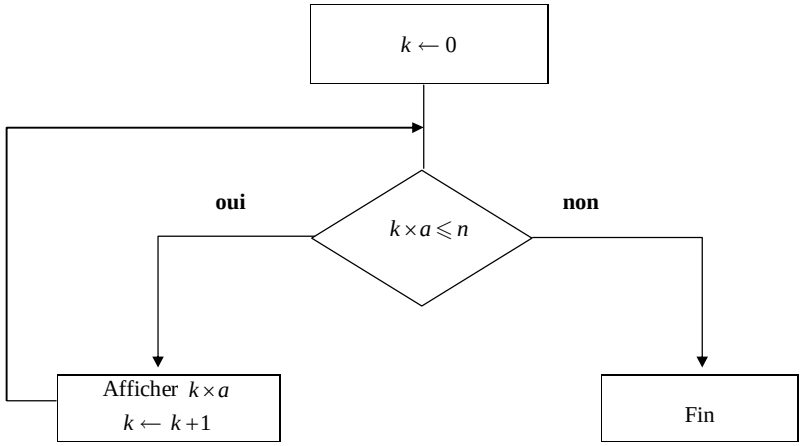
Fonction Python d'en tête `def mult(a,n) :` qui prend pour arguments deux entiers naturels *a* et *n* et qui renvoie tous les multiples de *a* inférieurs ou égaux à *n* :

L'écriture générique d'un multiple de *a* est *ka* avec $k \in \mathbb{Z}$.

Utilisation d'une boucle while :

```
def mult(a,n):
    k=0
    while k*a<=n:
          print(k*a)
          k=k+1
```

On peut faire un organigramme.



Version liste :

```
def mult(a,n):
    L=[]
    k=0
    while k*a<=n:
          L.append(k*a)
          k=k+1
    return L
```

Applications :

Fonction qui renvoie le plus grand multiple de *a* inférieur ou égal à *n* (cas particulier de la liste)

Fonction qui renvoie le quotient et le reste de la division euclidienne de *n* par *a* (lorsque *a* est non nul)

Passage de « la base dix à la base dix »

On donne les chiffres de l'écriture en base dix d'un entier naturel sous la forme d'une liste.

Exemple : `L=[5,3,1]`

On cherche à écrire un programme qui affiche le nombre en écriture de base 10.

Correctif : On donne une liste de chiffres. On désire écrire une fonction Python d'en-tête `def eebd(L) :` qui prend pour argument une liste L non vide de chiffres (compris entre 0 et 9) et qui renvoie l'entier naturel dont les chiffres de l'écriture en base 10 sont les éléments de L pris dans l'ordre de la liste.
On utilise l'écriture en base dix. On note une boucle « Pour » :

```
def eebd(L):
    N=0
    for i in range (len(L)):
       N=N+L[i]*10**(len(L)-i-1)
    return N
```

Pour une base *b*, on peut remplacer le 10 dans la boucle par *b*.

Amélioration : utilisation de la méthode de Horner.

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

Signification (ce que signifie extraire les chiffres de l'écriture en base dix d'un nombre)

Principe : On utilise des divisions euclidiennes par 10.

Lemme :

Soit n un entier naturel.
- Le chiffre des unités dans l'écriture en base dix de n est égal au reste de la division euclidienne de n par 10.
- Le nombre de dizaines de n est égal au quotient de la division euclidienne de n par 10.

Exemple :

division euclidienne de 547 par 10
$q = 54$ et $r = 7$

Complément :

Si on veut « casser » (« séparer » un nombre en deux), on choisit la puissance de 10 la plus adéquate.

Exemple : « Casser » 71352 en 713 et 52.

On divise par 100 (10^2).

$q = 713$ et $r = 52$

Remarque : La puissance doit correspondre à la base du nombre.

Le 20 décembre 2022

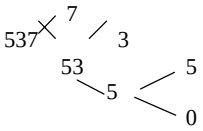
On se base sur le principe d'écriture d'un entier naturel dans une base quelconque.

```
def chiffres(n):
    while n>0:
          print(n%10)
          n=n//10
```

Commentaires :
1. On peut donner un autre nom à la fonction, par exemple `extract(n)`.
2. On ne peut pas échanger les instructions `print(n%10)` et `n=n//10`.
3. On peut remplacer la condition `n>0` par `n!=0`.


```
def chiffres(n):
    L=[]
    while n>0:
          L.append(n%10)
          n=n//10
    return L
```

Présentation en arbre



Le 15 décembre 2022 soir

Extraction des chiffres

Principe : division euclidienne par 10

$\underbrace{54}_{\text{quotient}}\ \underbrace{7}_{\text{reste de la division euclidienne par 10}}$

Deux derniers chiffres

$\underbrace{7137}\ \underbrace{52}$

Quotient reste de la division euclidienne par 100

permet de découper des nombres

Bases de Python

!= différent

! à ne pas confondre avec la notation de la factorielle d'un entier naturel

Le 15 décembre 2022 matin

n!=0

!= différent pas factorielle

Le 15 décembre 2022

conversion base b à base en 10 Python Frédéric Peurière

Le mercredi 4-1-2023

315 = 315 5
// 10 %10

return n//10

Le jeudi 7-11-2019

TS spé

Script Python extrait d'un extrait de livre « différentes bibliothèques »

```
def divisible(x,n):
    if x%n==0 :
       return True
    else:
       return False
```

Il s'agit d'une fonction qui a deux arguments.
Elle est utilisable dans d'autres programmes plus compliqués.
On a un type booléen.

On parle de fonction booléenne : fonction qui renvoie une variable de type booléen True ou False.

George Boole : mathématicien de la fin du XIX^{e} siècle.

Le 5-12-2020

a et b sont deux entiers naturels non nuls tels que $a > b$.

On ne veut utiliser que des additions. Pas de multiplications ni de soustractions.

On peut faire un petit schéma illustrant les bonds.

Fonction test_multiple(a, b)
N ← b
Tantque N < a **Faire**
N ← N + b
FinTantque
Si N = a
Afficher « a est un multiple de b »
Sinon
Afficher « a n'est pas un multiple de b »
FinSi
FinFonction

Dans la boucle « Tantque », on cherche le plus petit multiple de b supérieur ou égal à a.

```
def test_multiple(a,b):
    N=b
    while N<a:
          N=N+b
    if N==a:
       print("a est un multiple de b")
    else:
       print("a n'est pas un multiple de b")
```

Le 19 décembre 2022

Programme autour des écritures en base dix

```
def eebd(L):
    N=0
    l=len(L)
    for i in range(l):
        N=N+L[i]*10**(l-i-1)
    return N
```

On peut remplacer 10 par une base b quelconque avec b entier naturel supérieur ou égal à 2.
Méthode de Ruffini-Horner (voir article Wikipedia)

Opérations successives

http://lycee.stanislas.info.free.fr/Algorithme/Introduction/Horner.html réécriture de Horner d'un polynôme

Partie A
Partie entière, partie décimale d'un réel

I. Partie entière d'un réel

La notion de partie entière a déjà été rencontrée pour des simulations d'expériences aléatoires sur calculatrice ou tableur.

1°) Définition

La **partie entière d'un réel** x est l'unique entier relatif p tel que $p \leqslant x < p+1$.

2°) Notation

La partie entière d'un réel x est notée $\mathrm{E}(x)$.

On a donc : $\mathrm{E}(x) \leqslant x < \mathrm{E}(x)+1$.

/!\ Remarque importante :
L'inégalité de gauche est large $(\leqslant)$; l'inégalité de droite est stricte $(<)$.
Il faut donc bien faire attention à apprendre correctement la définition.

3°) Autres façons de définir la partie entière d'un réel

- La partie entière d'un réel x est le plus grand entier relatif inférieur ou égal à x.
- La partie entière d'un nombre réel est sa valeur décimale approchée par défaut à l'unité.

4°) Exemples

$\mathrm{E}(2,5)=2$ $\quad \mathrm{E}(10,9)=10$ $\quad \mathrm{E}(3)=3$ $\quad \mathrm{E}(-1,3)=-2$ $\quad \mathrm{E}(-5)=-5$ $\quad \mathrm{E}(-4,3)=-5$

On justifie toutes ces égalités par des encadrements :

$2 \leqslant 2,5 < 3$ $\quad 10 \leqslant 10,9 < 11$ $\quad 3 \leqslant 3 < 4$ $\quad -2 \leqslant -1,3 < -1$ $\quad -5 \leqslant -5 < -4$ $\quad -5 \leqslant -4,3 < -4$.

La partie entière d'un nombre décimal positif est égale à sa troncature à l'unité.

5°) Propriété

$\forall x \in \mathbb{Z} \quad \mathrm{E}(x) = x$

6°) Calculatrice

Une commande de la calculatrice permet d'obtenir la partie entière d'un réel.

Pour TI : [math] → NUM (ou NBRE) → choix 5

Sur TI-83 Plus : partEnt(ou int(

Sur TI-82 Stats : iPart(

Il faut se méfier du choix 3 (ent( ) qui ne correspond pas à la partie entière.

Pour CASIO : OPTN-NUM-Intg

II. Partie décimale d'un réel

1°) Définition

On appelle **partie décimale (ou fractionnaire)** d'un réel x le nombre $x - \mathrm{E}(x)$.

Ce nombre est noté $\mathrm{F}(x)$.

2°) Exemples

La partie décimale de 5,83 est égale à 0,83.

La partie décimale de – 5,4 est égale à 0,6.

3°) Propriétés immédiates

- On a toujours $\mathrm{E}(x)+\mathrm{F}(x)=x$ (la somme de la partie entière d'un nombre et de sa partie décimale est égale à ce nombre : partie entière d'un nombre + partie décimale = le nombre).

- Comme $\mathrm{E}(x) \leqslant x < \mathrm{E}(x)+1$, on a : $0 \leqslant x - \mathrm{E}(x) < 1$ soit $0 \leqslant \mathrm{F}(x) < 1$.

La partie décimale d'un réel est toujours comprise entre 0 (au sens large) et 1 (au sens strict).

4°) Mise en garde

- En 6e, on a défini la partie décimale d'un nombre décimal positif (partie du nombre à droite de la virgule). L'expression « partie décimale » a été employée dans un sens légèrement différent de celui défini ici.

- La calculatrice possède une fonction « partie décimale ».

Sur TI-83 Plus, la « partie décimale » est notée partDéc(et s'obtient ainsi : [math] → NUM → 4 : PartDéc(.
Sur TI-82 stats, la « partie décimale » est notée *f*Part(et s'obtient de la même façon.

Il faut se méfier de la fonction « partie décimale » de la calculatrice.
Pour les nombres positifs, le résultat affiché correspond à la définition que nous avons donnée.
Pour les nombres négatifs, le résultat affiché ne correspond pas à la définition.

Par exemple, la partie décimale affichée par la calculatrice de – 4,3 est – 0,3 et non 0,7.
La fonction « partie décimale » de la calculatrice ne correspond pas à notre définition.
Malheureusement la calculatrice ne possède pas de fonction qui corresponde à la définition mathématique de la partie décimale.

III. Test de divisibilité dans un programme

1°) Caractérisation des entiers relatifs à l'aide de la partie entière et de la partie décimale

Il s'agit d'une propriété caractéristique.

$x \in \mathbb{Z} \Leftrightarrow \mathrm{E}(x) = x$

$x \in \mathbb{Z} \Leftrightarrow \mathrm{F}(x) = 0$

2°) Application au test pour savoir si un nombre est un entier dans un programme

On utilise la fonction « partie entière » ou fonction « partie décimale » de la calculatrice.

Exemple : algorithme permettant de savoir si un entier naturel saisi en entrée est un carré parfait.

Le 7 septembre 2021

Jean Pastor-Menet

```
from math import *

def isent(n) :
    if float(floor(n)) = = float(n):
        return True
```

3°) Application au test de divisibilité dans un programme

La propriété de caractérisation sert à élaborer des tests dans des programmes : « test de divisibilité » (voir suite du chapitre).
Les fonctions partie entière et partie décimale permettent de rédiger facilement des conditions.

Considérons un entier relatif a et un entier relatif b non nul.

$b \mid a \Leftrightarrow \dfrac{a}{b}$ est un entier

$\Leftrightarrow \mathrm{E}\left(\dfrac{a}{b}\right) = \dfrac{a}{b}$

$\Leftrightarrow \mathrm{F}\left(\dfrac{a}{b}\right) = 0$

On retiendra le point suivant :

On peut utiliser $\mathrm{E}\left(\dfrac{a}{b}\right) = \dfrac{a}{b}$ pour traduire que $b \mid a$ selon le matériel de programmation utilisé.

Cas particulier important : test de parité

Considérons un entier relatif a.

a est pair $\Leftrightarrow 2 \mid a$

$\Leftrightarrow \dfrac{a}{2}$ est un entier

$\Leftrightarrow \mathrm{E}\left(\dfrac{a}{2}\right) = \dfrac{a}{2}$

4°) Autre test de divisibilité possible dans un programme

On doit aussi garder en mémoire la propriété suivante, qui est fondamentale, valable pour $a \in \mathbb{Z}$ et $b \in \mathbb{Z}^*$.

$b \mid a \Leftrightarrow$ le reste de la division euclidienne de a par b est égal à 0.

Cette propriété est utile pour un test de divisibilité dans un programme dans le cas où l'on a une commande de reste de division euclidienne.

Cas particulier : test de parité

a est pair $\Leftrightarrow$ le reste de la division euclidienne de a par 2 est égal à 0

Partie B
Division euclidienne et calculatrice

I. Rappels de propriétés et expression du quotient à l'aide de la partie entière

1°) Propriétés

• Propriété 1 [division euclidienne d'un entier naturel par un entier naturel non nul]

$\forall (a\,;b) \in \mathbb{N} \times \mathbb{N}^* \quad \exists ! (q\,;r) \in \mathbb{N} \times \mathbb{N} \;/\; a = bq + r \text{ et } 0 \leqslant r < b\,.$

• Propriété 2 [division euclidienne d'un entier relatif par un entier naturel non nul]

$\forall (a\,;b) \in \mathbb{Z} \times \mathbb{N}^* \quad \exists ! (q\,;r) \in \mathbb{Z} \times \mathbb{N} \;/\; a = bq + r \text{ et } 0 \leqslant r < b\,.$

2°) Expression du quotient à l'aide de la partie entière

• Propriété importante

a est un entier relatif et b est un entier naturel non nul.
Le quotient q de la division euclidienne de a par b est égal à la partie entière de $\frac{a}{b}$.

Autrement dit, avec la notation de la partie entière, on a: $\boldsymbol{q = \mathrm{E}\left(\frac{a}{b}\right)}$.

• Démonstration

En reprenant la démonstration de l'existence du couple (q, r), on a défini q comme le plus grand entier tel que $qb \leqslant a$ soit, comme $b > 0$, $q \leqslant \frac{a}{b}$.

• Utilisation

Cette expression sert assez rarement en pratique mais peut servir en programmation.

4°) Expression du reste à l'aide de la partie entière et de la partie décimale

• Expression du reste à l'aide de la partie entière

L'égalité $a = bq + r$ permet d'écrire $r = a - bq$.
Le reste donc est égal à $a - bq$.

D'après le paragraphe 3°), on a donc $\boldsymbol{r = a - b \times \mathrm{E}\left(\frac{a}{b}\right)}$.

Il faut connaître cette expression sinon par cœur du moins suffisamment bien pour pouvoir la retrouver rapidement.

• Expression du reste à l'aide de la partie fractionnaire

D'après l'expression précédente, on peut écrire $r = b \times \left[\frac{a}{b} - \mathrm{E}\left(\frac{a}{b}\right)\right]$.

On a donc $\boldsymbol{r = b \times \mathrm{F}\left(\frac{a}{b}\right)}$.

• Utilisation

Ces expressions servent assez rarement en pratique mais peuvent servir en programmation.

4°) Bilan

Dans la division euclidienne d'un entier relatif a est un entier relatif par un entier naturel non nul b,
le quotient est donné par : $\boldsymbol{q = \mathrm{E}\left(\frac{a}{b}\right)}$;
le reste est donné par $\boldsymbol{r = a - b \times \mathrm{E}\left(\frac{a}{b}\right)}$ ou $\boldsymbol{r = b \times \mathrm{F}\left(\frac{a}{b}\right)}$.

II. Effectuer une division euclidienne à l'aide d'une calculatrice

1°) Calculatrice de collège (pour mémoire)

La calculatrice Casio fx-92 possède une touche de division euclidienne ⊢ qui marche pour deux entiers positifs mais qui ne marche pas lorsque l'un des deux entiers est négatif.

Exemple :

356 ⊢ 17

Q = 20 ; R = 16

Attention -356 ⊢ 17

$-\frac{356}{17}$ ou $-10{,}94117647\ldots$

2°) Calculatrice de lycée

• Calculatrice TI-83 Plus.fr (noire) ou TI-83 Premium CE

On peut obtenir le reste de la division euclidienne d'un entier naturel par un entier naturel non nul.

Appuyer sur la touche math ; sélectionner NUM ou NBRE puis choisir 0 : remainder(ou 0 : reste(

: remainder(75,4 ou : reste(75,4 donne le reste de la division euclidienne de 75 par 4.

Les deux nombres doivent être séparés par une virgule.

On obtient uniquement le reste et pas le quotient.
Le quotient s'obtient facilement. En effet, l'égalité de la division euclidienne d'un entier naturel a par un entier naturel b non nul s'écrit $a = bq + r$ où q désigne le quotient et r le reste. On obtient $q = \frac{a - r}{b}$.
Attention, cette commande ne marche que pour des entiers positifs.

• Calculatrice TI-83

On ne peut pas obtenir la division euclidienne d'un entier par un autre.
On est obligé d'utiliser une division décimale.

Exemple :

Effectuer la division euclidienne de 169 204 par 37 en utilisant la calculatrice.

① $169\,204 : 37 = 4\,573,081\,08\ldots$ (il s'agit d'une division décimale)
La partie entière est égale à 4 573.

② $4\,573 \times 37 = 169\,201$

③ Le reste est égal à $169\,204 - 169\,201 = 3$.

169 204	37
– 169 201	4 573
3	

On peut écrire l'égalité de la division euc1idienne : $169\,204 = 37 \times 4\,573 + 3$.

Remarques :

• On peut aussi directement calculer la partie entière de $\frac{169\,204}{37}$.

• Cette méthode permet d'effectuer la division euclidienne d'entiers positifs ou négatifs par un entier naturel non nul.

III. Algorithme de division euclidienne d'un entier relatif par un entier naturel non nul

1°) Algorithme en langage naturel

L'algorithme suivant demande en entrée un entier relatif a et un entier naturel $b \neq 0$ et affiche en sortie le quotient q et le reste r de la division euclidienne de a par b.
Les variables a, b, q, r sont donc toutes des entiers.

Entrées :
Saisir a
Saisir b

Traitement :
r prend la valeur du reste de la division euclidienne de a par b
q prend la valeur de $\frac{a-r}{b}$

Sorties :
Afficher q
Afficher r

Entrées :
Saisir a
Saisir b

Traitement :
q prend la valeur $\mathrm{E}\left(\frac{a}{b}\right)$
r prend la valeur $a - bq$

Sorties :
Afficher q
Afficher r

2°) Programme sur calculatrice TI

▪ Pour tous les modèles TI :

Pour obtenir le quotient et le reste la division euclidienne d'un entier *relatif* A par un entier naturel B non nul, on peut utiliser la partie entière (partEnt).

Le quotient de la division euclidienne d'un entier relatif A par un entier naturel B non nul peut être obtenu par partEnt(A/B) ou int(A/B).

Le reste de la division euclidienne d'un entier relatif A par un entier naturel B non nul peut être obtenu par A – partEnt(A/B)∗B ou A – int(A/B)∗B .

▪ Pour les modèles TI les plus récents :

Pour obtenir le quotient et le reste la division euclidienne d'un entier *naturel* A par un entier naturel B non nul, on peut utiliser la fonction remainder ou reste qui ne fonctionne qu'avec des entiers positifs.

Le reste de la division euclidienne d'un entier *naturel* A par un entier naturel B non nul peut être obtenu par remainder(A,B) ou reste(A,B).

Le quotient de la division euclidienne d'un entier naturel A par un entier naturel B non nul peut être obtenu par (A – remainder(A,B))/ B ou (A – reste(A,B))/ B.

• Premier programme sur TI :

Programme 1 : $A \in \mathbb{N}$, $B \in \mathbb{N}^*$; Programmes 2 et 3 : $A \in \mathbb{Z}$, $B \in \mathbb{N}^*$

```
: Prompt A, B
: remainder(A,B) → R
: (A – R)/B → Q
: Disp Q, R
```

```
: Prompt A, B
: partEnt(A/B) → Q
: A – Q×B → R
: Disp Q, R
```

```
: Prompt A, B
: partEnt(A/B) → Q
: remainder(A,B) → R
: Disp Q, R
```

partEnt à remplacer éventuellement par int (int(A/B)

• Deuxième programme sur TI avec quelques raffinements concernant l'affichage :

```
PROGRAM : DIVEUCLI
: EffEcr
: Disp "A = QB + R"
: Input "A =", A
: Input "B =", B
: remainder(A,B) → R
: (A – R) / B → Q
: Disp "Q =", Q
: Output(5,3,Q)
: Disp "R =", R
: Output(6,3,R)
```

```
PROGRAM : DIVEUCLI
: EffEcr
: Disp "A = QB + R"
: Input "A =", A
: Input "B =", B
: partEnt(A/B) → Q   ou int(A/B) → Q
: (A – (Q * B)) → R
: Disp "Q =", Q
: Output(5,3,Q)
: Disp "R =", R
: Output(6,3,R)
```

Sur les modèles en anglais, remplacer EffEcr (effacer l'écran) par ClrHome.

IV. Algorithme de division euclidienne d'un entier relatif par un entier relatif non nul

1°) Algorithme en langage naturel

L'algorithme suivant demande en entrée deux entiers relatifs a et b $(b \neq 0)$ et affiche en sortie le quotient q et le reste r de la division euclidienne de a par b.
Les variables a, b, q, r sont donc des entiers.

Entrées :
Saisir a
Saisir b

Traitement :
Si $b > 0$
  Alors q prend la valeur $\mathrm{E}\left(\frac{a}{b}\right)$
  Sinon q prend la valeur $-\mathrm{E}\left(-\frac{a}{b}\right)$
FinSi
r prend la valeur $a - bq$

Sorties :
Afficher q
Afficher r

2°) Programmation

La programmation ne pose pas de difficultés.

Partie C
Liste des diviseurs positifs d'un entier naturel non nul

I. Un premier algorithme

1°) Problème

Faire « à la main » la liste des diviseurs positifs d'un entier naturel non nul peut s'avérer extrêmement fastidieux.
Le recours à un programme permet de gagner beaucoup de temps.

Le but de ce paragraphe est d'écrire un algorithme qui affiche en sortie les diviseurs positifs d'un entier naturel n non nul saisi en entrée.

Rappelons que, comme nous nous intéressons à un entier naturel $n \geqslant 1$, on peut aussi bien dire « diviseurs strictement positifs » que « diviseurs positifs ou nuls » puisque 0 n'est pas un diviseur de n.

On rentrera sur calculatrice le programme « optimal » donné dans le paragraphe **III**.

2°) Algorithme

On veut rédiger un algorithme en langage naturel qui, pour un entier naturel $n \geqslant 1$ saisi en entrée, affiche en sortie tous ses diviseurs positifs.
On utilise une boucle.

Le principe consiste à tester tous les entiers de 1 à n pour savoir si ce sont des diviseurs de n.
En effet, on sait que les diviseurs positifs de n sont inférieurs ou égaux à n. On peut donc commencer à 1 et s'arrêter à n.

On connaît le nombre d'itérations (ici n) ; une boucle « Pour » est donc bien adaptée.

On va écrire un premier algorithme sans se préoccuper de son « coût ».

Entrée :
Saisir n (entier naturel non nul)

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

Il est demandé de savoir réécrire cet algorithme.

Le 23-1-2020

Programme Python

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

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

Version encore plus courte en syntaxe : utilisation de listes en compréhension :

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

3°) Programme sur calculatrice TI

Suivant le modèle de calculatrice utilisé, on peut exprimer la divisibilité soit à l'aide de la commande de reste de division euclidienne soit à l'aide de la partie entière ou de la partie décimale.

Modèles TI récents

```
: Prompt N
: For(I,1,N)
: If remainder(N, I) = 0
: Then
: Disp I
: Pause
: End
: End
```

Modèles TI anciens

```
: Prompt N
: For(I,1,N)
: If ent(N / I) = N / I   [ou int(N / I) = N / I ]
: Then
: Disp I
: Pause
: End
: End
```

Attention, à bien taper la lettre I et non le chiffre 1. On peut éventuellement changer la lettre I en K.

4°) Commentaire

Cet algorithme n'est pas optimal. Il va s'avérer extrêmement lent quand on va le « passer » sur machine.

II. Amélioration de l'algorithme

1°) Remarque

Les diviseurs positifs d'un entier naturel non nul « fonctionnent » par paires.
On va donc pouvoir écrire un algorithme qui va beaucoup plus vite car il va permettre de diminuer les calculs.
Pour cela, on va utiliser un lemme.

2°) Lemme

● Énoncé

n est un entier naturel supérieur ou égal à 1.

> Soit d et d' deux diviseurs positifs associés de n (on a donc $dd' = n$).
> Le plus petit de ces deux diviseurs est inférieur ou égal à $\sqrt{n}$; le plus grand est supérieur ou égal à $\sqrt{n}$.

● Démonstration (à savoir refaire)

1ère méthode :

On réalise un raisonnement par l'absurde.

Supposons que $d > \sqrt{n}$ et $d' > \sqrt{n}$.
En multipliant membre à membre ces deux inégalités qui ne comportent que des nombres strictement positifs, on obtient : $dd' > n$ soit $n > n$ ce qui est absurde.

2e méthode :

• 1er cas : $d \leqslant \sqrt{n}$
Dans ce cas, la propriété est démontrée.

• 2e cas : $d > \sqrt{n}$ (1)
Dans ce cas, $d' = \dfrac{n}{d}$.

D'après (1), $\dfrac{1}{d} < \dfrac{1}{\sqrt{n}}$ d'où $\dfrac{n}{d} < \dfrac{n}{\sqrt{n}}$ soit $d' < \sqrt{n}$ (car $\dfrac{n}{\sqrt{n}} = \dfrac{\sqrt{n} \times \sqrt{n}}{\sqrt{n}} = \sqrt{n}$).

La propriété est donc démontrée.

3°) Algorithme amélioré

n est un entier naturel non nul.

Entrée :
Saisir *n*

Traitement et sorties :
Pour *i* allant de 1 à $E\left(\sqrt{n}\right)$ **Faire**
  Si *i* est un diviseur de *n* (ou **Si** *i* | *n*)
    Alors afficher *i* et $\frac{n}{i}$
  FinSi
FinPour

On peut aisément programmer cet algorithme sur calculatrice.

III. Nouvelle amélioration

1°) Objectif

On peut améliorer l'algorithme pour éviter qu'il affiche les « doublets ». Attention, cela pourrait avoir une incidence si l'on ajoutait une instruction permettant de donner le nombre de diviseurs de l'entier *n* saisi en entrée (cela rajouterait 1 au résultat).
La situation d'apparition d'un doublet n'a lieu que dans le cas où l'entier *n* saisi en entrée est un carré parfait.

2°) Algorithme

n est un entier naturel non nul.

Entrée :
Saisir *n*

Traitement et sorties :
Pour *i* allant de 1 à $E\left(\sqrt{n}\right)$ **Faire**
  Si *i* est un diviseur de *n* (ou **Si** *i* | *n*)
    Alors
      Si $i \neq \frac{n}{i}$ (ou **Si** $i^2 \neq n$)
        Alors afficher *i* et $\frac{n}{i}$
        Sinon afficher *i*
      FinSi
  FinSi
FinPour

3°) Programme sur calculatrice TI pour éviter les doublets

```
: Prompt N
: For(I,1,√(N))    [ou  For(I,1,√(N))]
: If remainder(N,I)=0    [ou ent(N/I)=N/I ou int(N/I)=N/I]
: Then
: If I≠N/I        [ou I²≠N : ce qui représente un léger gain de temps]
: Then
: Disp I, N/I
: Pause
: Else
: Disp I
: End
: End
: End
```

• On peut constater que ce programme est beaucoup plus rapide. Par exemple, pour 360, c'est pratiquement immédiat.

• À cause du « Pause », il faut appuyer sur entrer à chaque fois pour voir afficher les diviseurs au fur et à mesure (affichage par groupes de 2 en général pour tous les diviseurs ayant un diviseur associé différent).

• La calculatrice affiche les diviseurs entiers naturels par paires.

• Comme on le voit, on n'est pas obligé d'utiliser la partie entière de $\sqrt{N}$.

• On est obligé d'écrire les 3 « End » (les End fonctionnent comme des parenthèses, un seul End ne suffit).

IV. Utilisation de listes

1°) Algorithme

Un programme par liste permet d'avoir un affichage plus clair.

On reprend l'algorithme du **I** en utilisant une liste L supposée créée et vide avant le début de l'algorithme.

Version 1 : version non opérationnelle pour la programmation [à savoir réécrire parfaitement]

Version 2 : version opérationnelle de l'algorithme précédent pour la programmation sur calculatrice

Les variables de la version 1 sont :
• *n* et *i* qui sont des entiers naturels ;
• L qui est une liste.

Les variables de la version 2 sont :
• *n*, *k* et *i* qui sont des entiers naturels ;
• L qui est une liste.

Version 1

Entrée :
Saisir n

Traitement :
Pour i allant de 1 à n **Faire**
  Si $i \mid n$
    Alors ajouter i à la liste L
  FinSi
FinPour

Sortie :
Afficher L

Version 2

Entrée :
Saisir n

Initialisation :
k prend la valeur 1

Traitement :
Pour i allant de 1 à n **Faire**
  Si $i \mid n$
    Alors $L(k)$ prend la valeur i
      k prend la valeur $k+1$
  FinSi
FinPour

Sortie :
Afficher L

• Il est demandé de savoir réécrire cet algorithme.

• L'instruction « k prend la valeur 1 » tient compte du fait que la numérotation des éléments d'une liste sur calculatrice démarre à 1 et non à 0.

• Dans la liste L affichée en sortie, les diviseurs seront donnés dans l'ordre croissant.

2°) Programme correspondant sur calculatrice TI

```
: Prompt N
: 1 → K
: ClrList L1
: For (I,1,N)
: If remainder(N, I) = 0
: Then
: I → L1(K)
: K+1 → K
: End
: End
: Disp L1
```

Indications et remarques :

• Pour obtenir l'instruction ClrList ou EffListe, il faut appuyer sur la touche stats puis aller dans EDIT et choisir 4 : .

• Pour écrire L1 dans le programme, appuyer sur les touches 2nde 1 .

• À la fin du programme, on peut remplacer la dernière instruction « Disp L1 » par « Pause L1 ». On doit alors appuyer sur la touche entrer pour voir les éléments de L1 c'est-à-dire les diviseurs positifs de N.

• Pour les calculatrices TI-83 plus, mettre $K+1$ entre parenthèses (sinon il peut y avoir des problèmes quand on fait tourner la calculatrice pour certaines valeurs de N, par exemple pour $N=100$.

• Pour faire défiler la liste des diviseurs positifs (utile lorsqu'il y en a beaucoup) :
2nde 1 (L1) → entrer puis faire défiler avec la flèche (touche du clavier).

Autre méthode : 2nde stats (listes) → entrer puis faire défiler avec la flèche (touche du clavier)

• À la fin du programme, on peut remplacer la dernière instruction « Disp L1 » par « Pause L1 ». On doit alors appuyer sur la touche entrer pour voir les éléments de L1 c'est-à-dire les diviseurs.

• Ce programme est très utile pour les exercices.

3°) Algorithme évitant les doublets

Version 1 : version non opérationnelle pour la programmation

Version 2 : version opérationnelle de l'algorithme précédent pour la programmation sur calculatrice

Version 1

Entrée :
Saisir n

Traitement :
Pour i allant de 1 à $E(\sqrt{n})-1$ **Faire**
  Si $i \mid n$
    Alors ajouter i et $\frac{n}{i}$ à la liste L
  FinSi
FinPour
Si $\sqrt{n}$ est un entier
  Alors ajouter $\sqrt{n}$ à la liste L
FinSi

Sortie :
Afficher L

Version 2

Entrée :
Saisir n

Initialisation :
k prend la valeur 1

Traitement :
Pour i allant de 1 à $E(\sqrt{n})-1$ **Faire**
  Si $i \mid n$
    Alors $L(k)$ prend la valeur i
      $L(k+1)$ prend la valeur $\frac{n}{i}$
      k prend la valeur $k+2$
  FinSi
FinPour
Si $E(\sqrt{n}) = \sqrt{n}$
  Alors $L(k)$ prend la valeur $\sqrt{n}$
FinSi

Sortie :
Afficher L

La programmation sur calculatrice ne pose pas de problème.

Il est possible de rajouter à la fin du programme une instruction permettant d'ordonner les éléments de la liste des diviseurs dans l'ordre croissant avant de l'afficher.

Le 13 octobre 2016

On peut ordonner la liste des diviseurs.

Partie D
La division euclidienne « à l'ancienne »

Dans toute cette partie, on se place dans le cas de la division euclidienne de deux entiers naturels.

I. Méthode des soustractions successives

1°) Introduction

Les premiers processeurs ne faisaient que des additions et des soustractions ! Comment peut-on réaliser une division euclidienne de deux entiers naturels ?

2°) Méthode

La méthode naturelle employée pour faire la division euclidienne d'un entier naturel a par un entier naturel b non nul procède par soustractions successives.

On soustrait b de a tant que c'est possible et on compte le nombre de soustractions faites.

C'est la méthode décrite par Euclide. Cette méthode peut s'illustrer géométriquement avec une règle de longueur a.

3°) Exemple

On cherche le quotient et le reste de la division euclidienne de 11 par 3.

On calcule $11-3=8$ et on compte 1
puis $8-3=5$ et on compte 2
puis $5-3=2$ et on compte 3
puis $2-3$ qui est strictement négatif donc on s'arrête de compter !

Le reste et le quotient de la division euclidienne de 11 par 3 sont respectivement 2 et 3 (on a : $11=3\times3+2$).
Cette méthode est algorithmique ; on peut donc facilement écrire un algorithme. On peut ensuite réaliser le programme correspondant.

4°) Interprétation géométrique

a et b sont deux entiers naturels, avec b non nul.

• On dispose d'un segment de longueur a.

• Tant que le segment est de longueur supérieure ou égale à b, on coupe une longueur b de la partie de segment restante.

• La longueur finale est le reste de la division euclidienne de a par b.

II. Algorithme de division euclidienne correspondant à la méthode des soustractions successives

1°) Algorithme rédigé en langage intermédiaire

L'algorithme demande d'entrer un entier naturel a et un entier naturel b non nul.

À arranger (méthode pour obtenir le reste par des différences successives) :

- Effectuer la soustraction euclidienne de a par b et noter d la différence ;
- Remplacer a par b ;
- Remplacer b par r ;
- Recommencer les calculs précédents jusqu'à ce qu'une division donne un reste égal à 0.
- Le reste est le …

2°) Algorithme rédigé en langage naturel (langage structuré)

L'algorithme demande d'entrer un entier naturel a et un entier naturel b non nul.

On utilise une boucle « Tantque » (boucle avec test d'arrêt).

Entrée :
Saisir a et b

Initialisations :
q prend la valeur 0
m prend la valeur a

Traitement :
Tantque $m \geqslant b$ **Faire**
    q prend la valeur $q+1$
    m prend la valeur $m-b$
FinTantque

Sortie :
Afficher q et m

Commentaires :

• L'algorithme utilise 4 variables (a, b, q, m) qui sont toutes des entiers naturels.
Les valeurs de q et m affichées en sortie sont respectivement le quotient et le reste de la division euclidienne de a par b.

• Comme pour tout algorithme avec une boucle « Tantque », il est possible de faire l'organigramme.

• Comme $b>0$, la condition $m \geqslant b$ est fausse en un nombre fini d'étapes. Autrement dit, on n'a pas de boucle infinie.

• On peut regarder ce qui se passe si les entiers naturels a et b en entrée sont tels que $a < b$.
Dans ce cas, la condition $m \geqslant b$ est fausse dès le début.
Il n'y a donc pas de boucle.
Le résultat affiché pour q en sortie est 0 et le résultat affiché pour m en sortie est a, ce qui est correspond bien au quotient et au reste de la division euclidienne de a par b puisque $a = b \times 0 + a$.

• On pourra noter qu'à chaque étape, les variables vérifient la relation : $a = bq + m$.
On dit qu'il s'agit d'un « invariant de boucle ».
La notion d'invariant de boucle ne sera pas développée cette année. Elle est juste mentionnée à l'occasion de cet algorithme.
On pourra vérifier cette propriété dans l'exemple détaillé dans le paragraphe suivant.

• Cet algorithme est donné à titre de curiosité ; il n'a pas tellement d'intérêt.

3°) Exemple de fonctionnement « à la main »

On prend $a = 17$ et $b = 3$.
On dresse un tableau d'évolution des variables m et q permettant de dérouler l'algorithme pas à pas (les variables a et b n'évoluent pas au cours de l'algorithme).

Étape	**0**	**1**	**2**	**3**	**4**	**5**
q	0	1	2	3	4	5
m	17	14	11	8	5	2
Condition $m \geqslant b$	vraie	vraie	vraie	vraie	vraie	faux

Dans la division euclidienne de 17 par 3, le quotient est 5 et le reste est 2.

4°) Programme correspondant sur calculatrice

On peut réaliser le programme correspondant à cet algorithme sur calculatrice ou sur ordinateur.

La *performance du programme* est moindre, comme nous l'avons dit, qu'un programme direct sans boucle.
La boucle nécessite un certain nombre d'opérations qui vont prendre du temps.

```
: Prompt A, B
: 0 → Q
: A → M
: While M ⩾ B
: Q+1 → Q
: M – B → M
: End
: Disp Q
: Disp M
```


5°) Coût et performance de l'algorithme

Le coût d'un algorithme correspond au nombre total d'opérations effectués lors de l'exécution de l'algorithme.
Il correspond au temps d'exécution du programme (rapide ou lent) lorsque l'on passe sur « machine ».

Cet algorithme est évidemment plus long que l'algorithme qui consiste à calculer le quotient de a par b puis à prendre la partie entière car il y a beaucoup plus d'opérations.

Partie E
Retour sur l'ensemble des diviseurs positifs d'un entier naturel non nul

Dans l'amélioration de l'algorithme donnant la liste des diviseurs positifs d'un entier naturel non nul, nous avons démontré la propriété suivante :

> Soit d et d' deux diviseurs positifs associés d'un entier naturel $n \geqslant 1$.
> Alors l'un de ces diviseurs est inférieur ou égal à $\sqrt{n}$ et l'autre est supérieur ou égal à $\sqrt{n}$.

Cela résultat permet de préciser une propriété sur la structure et le nombre de diviseurs positifs d'un entier naturel $n \geqslant 1$.
On distingue deux cas selon que n est un carré parfait ou non.

Rappelons qu'un entier naturel est appelé « carré parfait » lorsque c'est le carré d'un entier (par exemple, 36 est un carré parfait).

1er cas : n n'est pas un carré parfait

Le nombre de diviseurs positifs de n est égal au double du nombre de diviseurs positifs de n inférieurs ou égaux (en fait strictement inférieurs puisque n est un carré parfait) à $\sqrt{n}$.
C'est donc un nombre pair.

Exemple :

$n = 18$

Les diviseurs positifs de 18 sont 1, 2, 3, 6, 9, 18. Il y en a 6 qui est bien un nombre pair.

Remarque :

Dans ce cas, on peut dire que les diviseurs positifs de n se « partagent » en deux groupes : les diviseurs strictement inférieurs à $\sqrt{n}$ et les diviseurs strictement supérieurs à $\sqrt{n}$.

2e cas : n est un carré parfait

Le nombre de diviseurs positifs de n est égal au double du nombre de diviseurs positifs de n strictement inférieurs à $\sqrt{n}$ auquel on ajoute 1 (puisque $\sqrt{n}$ est un diviseur positif de n).

C'est donc un nombre impair.

Exemple :

$n = 25$

Les diviseurs positifs de 25 sont 1, 5, 25. Il y en a 3 qui est bien un nombre impair.

Par disjonction de cas, on peut formuler la propriété pour un entier naturel $n \geqslant 1$:

> *n* est un carré parfait si et seulement si le nombre de diviseurs positifs de *n* est un nombre impair.

Partie F
Quelques formules utilisant la partie entière

I. Nombre de multiples strictement positifs d'un entier naturel non nul inférieurs ou égaux à un réel positif donné

1°) Propriété

> *a* est un entier naturel non nul donné.
> M est un réel positif ou nul donné.
> Le nombre de multiples de *a* strictement positifs inférieurs ou égaux à M est égal à $\mathrm{E}\left(\frac{\mathrm{M}}{a}\right)$.

2°) Démonstration

Les multiples de *a* sont les entiers relatifs de la forme *ka* avec $k \in \mathbb{Z}$.

$ka \leqslant \mathrm{M} \Leftrightarrow k \leqslant \frac{\mathrm{M}}{a}$

Donc les multiples de *a* strictement positifs inférieurs ou égaux à M sont les entiers de la forme *ka* où *k* est un entier naturel tel que $1 \leqslant k \leqslant \mathrm{E}\left(\frac{\mathrm{M}}{a}\right)$ d'où le résultat.

II. Nombre de carrés parfaits strictement positifs inférieurs ou égaux à un réel positif donné

1°) Propriété

> M est un réel positif ou nul donné.
> Le nombre de carrés parfaits strictement positifs inférieurs ou égaux à M est égal à $\mathrm{E}\left(\sqrt{\mathrm{M}}\right)$.

2°) Démonstration

Les carrés parfaits sont les entiers naturels de la forme a^2 avec $a \in \mathbb{N}$.

$a^2 \leqslant \mathrm{M} \Leftrightarrow a \leqslant \sqrt{\mathrm{M}}$

Donc carrés parfaits strictement positifs inférieurs ou égaux à M sont les entiers de la forme a^2 où *a* est un entier naturel tel que $1 \leqslant a \leqslant \mathrm{E}\left(\sqrt{\mathrm{M}}\right)$ d'où le résultat.

III. Nombre de chiffres de l'écriture en base dix d'un entier naturel

Le but de ce paragraphe est de trouver une formule donnant le nombre de chiffres d'un entier naturel.

Les nombres entiers naturels à 1 chiffre sont compris entre 10^0 (large) et 10^1 (strict).
Les nombres entiers naturels à 2 chiffres sont compris entre 10^1 (large) et 10^2 (strict).

….

Les nombres entiers naturels à *n* chiffres sont compris entre 10^{n-1} (large) et 10^n (strict).

Par exemple, $10^3 \leqslant 2317 < 10^4$.

On considère un entier naturel non nul A à *n* chiffres ($n \geqslant 1$).

On a : $10^{n-1} \leqslant \mathrm{A} < 10^n$ (1).

(1) donne $\log\left(10^{n-1}\right) \leqslant \log \mathrm{A} < \log 10^n$ car la fonction log est strictement croissante sur $\mathbb{R}_+^*$.

On a donc $n-1 \leqslant \log \mathrm{A} < n$ (2).
(2) permet de dire que $n-1 = \mathrm{E}(\log \mathrm{A})$ d'où $n = \mathrm{E}(\log \mathrm{A}) + 1$.
On retiendra le résultat suivant dont il demandé de retenir la démonstration :

> **Le nombre de chiffres en base 10 d'un entier naturel A non nul est égal à $\mathrm{E}(\log \mathrm{A}) + 1$.**

Exemples d'utilisation :

① $\mathrm{A} = 25^{40}$

$\log \mathrm{A} = 40 \log 25$

$= 55,9176\ldots$

Le nombre de chiffres de A est égal à $55 + 1 = 56$.

On aurait pu obtenir ce résultat avec la calculatrice.
En effet, avec la calculatrice, on obtient l'affichage suivant pour le calcul de A : $8.271806126 * 10^{55}$.

Il suffit d'ajouter 1 pour trouver qu'il y a 56 chiffres.

② $B = 2^{2013}$

$\log B = 2013 \log 2$

$= 605,9733\ldots$

Le nombre de chiffres de B est égal à $605+1=606$.

Pour B, il n'est pas possible d'utiliser la calculatrice pour trouver le nombre de chiffres de l'écriture en base 10 de B. Celle-ci est en dépassement de capacité.

Partie G
Algorithme d'extraction des chiffres

Il s'agit d'une procédure de base. On peut s'en servir dans d'autres situations.

On veut écrire un algorithme qui, pour un entier naturel *n* saisi en entrée, affiche en sortie les chiffres de son écriture en base 10.

Principe :

- On effectue la division euclidienne de *n* par 10, le reste obtenu est le chiffre des unités de *n* noté r_0.
- On effectue la division euclidienne de $n - r_0$ par 10.

Etc.

On va écrire un algorithme d'extraction des chiffres d'un entier naturel non nul dont l'écriture en base 10 comporte 3 chiffres.

Algorithme :

Entrée :
Saisir *n*

Traitement :
a prend la valeur du reste de la division euclidienne de *n* par 10

b prend la valeur du reste de la division euclidienne de $\dfrac{n-a}{10}$ par 10

c prend la valeur $\dfrac{n-a-10b}{10}$ par 10

Sortie :
Afficher *a*
Afficher *b*
Afficher *c*

Programme correspondant sur calculatrice :

```
: Prompt N
: remainder(N,10) → A
: remainder(N – A/10,10) → B
: remainder((N – A – 10B)/100,10) → C
: Disp A
: Disp B
: Disp C
```