*Thème: Arithmétique*

TI-83 Premium CE

# Diviseurs, nombres premiers et pgcd.

python

On va voir dans cette activité quelques fonctions classiques en arithmétique.

## Dans un script ARITHM

1°) Ecrire une fonction `divise` qui prend comme argument `a`$\in \mathbb{Z}^*$ et `b`$\in \mathbb{Z}$ et qui renvoie `True` si `a` divise `b` et `False` sinon.

```
PYTHON SHELL
>>> divise(5,7)
False
>>> divise(5,10)
True
```

2°) Ecrire une fonction `nbdiviseur` qui prend comme argument $n \in \mathbb{N}^*$ et qui renvoie le nombre de diviseurs positifs de $n$.

```
PYTHON SHELL
>>> nbdiviseur(10)
4
>>> nbdiviseur(11)
2
```

3°) Ecrire une fonction `listediviseur` qui prend comme argument $n \in \mathbb{N}^*$ et qui renvoie la liste des diviseurs positifs de $n$.

```
PYTHON SHELL
>>> listediviseur(10)
[1, 2, 5, 10]
>>> listediviseur(4569878)
[1, 2, 29, 58, 78791, 157582, 22
84939, 4569878]
```

4°) Ecrire une fonction `listedivcommun` qui prend comme argument $a$ et $b$ deux entiers naturels non nuls et qui renvoie la liste des diviseurs positifs communs à $a$ et $b$.

```
PYTHON SHELL
>>> listedivcommun(6,8)
[1, 2]
>>> listedivcommun(30,75)
[1, 3, 5, 15]
```

5°) Ecrire une fonction `pgcd` qui prend comme arguments $a$ et $b$ deux entiers naturels non nuls et qui renvoie $pgcd(a,b)$ en utilisant la fonction `listedivcommun`.

```
PYTHON SHELL
>>> pgcd(6,8)
2
>>> pgcd(30,75)
15
```

6°) Ecrire une fonction `ppcm` qui prend comme arguments $a$ et $b$ deux entiers naturels non nuls et qui renvoie $ppcm(a,b)$.

```
PYTHON SHELL
>>> ppcm(6,8)
24
>>> ppcm(30,75)
150
```

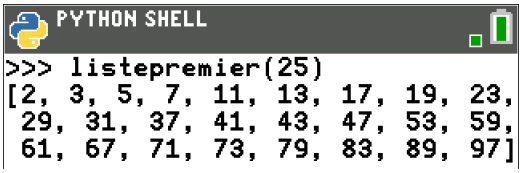
7°) Ecrire une fonction `premier` qui prend comme argument $n \in \mathbb{N}^*$ et qui renvoie `True` si $n$ est premier et `False` sinon.

```
PYTHON SHELL
>>> premier(1)
False
>>> premier(2)
True
>>> premier(10)
False
```

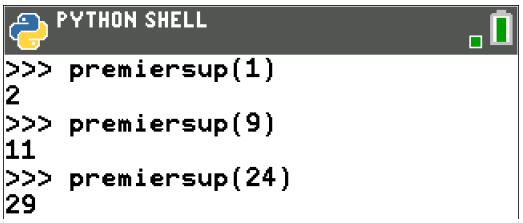
# Diviseurs, nombres premiers et pgcd.

8°) Ecrire une fonction `npremier` qui prend comme argument $n \in \mathbb{N}^*$ et qui renvoie le n-ième nombre premier.


9°) Ecrire une fonction `listepremier` qui prend comme argument $n \in \mathbb{N}^*$ et qui renvoie la liste des nombres premiers inférieurs ou égaux à $n$.



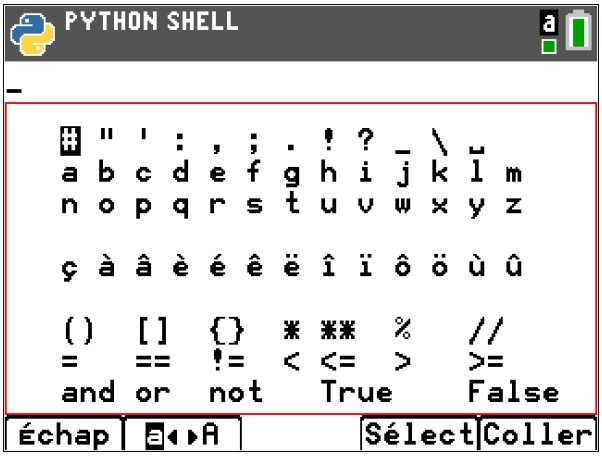

10°) Ecrire une fonction `premiersup` qui prend comme argument $n \in \mathbb{N}^*$ et qui renvoie le plus petit nombre premier supérieur strict à $n$.

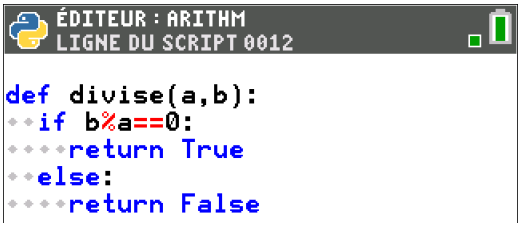



## Fonction `divise`

1°) Pour obtenir le reste de la division euclidienne de $b$ par $a$ on utilise le symbole %.

Il est accessible dans [a A #], puis à l'aide des flèches de direction de la calculatrice, on va chercher le symbole % qu'il faut sélectionner en appuyant sur [Sélect] puis sur [Coller] pour l'insérer dans le texte du script.




Attention : Pour faire le test si le reste est nul, il ne faut pas oublier le double égal == qui correspond à la comparaison (un seul = correspond à l'affectation).

*Thème: Arithmétique*

TI-83 Premium CE

# Diviseurs, nombres premiers et pgcd.

## Fonction `nbiviseur`

2°) Pour compter les diviseurs de l'entier `n`$\in \mathbb{N}^*$, on commence par introduire un compteur $k$ qu'on initialise à 1. En effet tous les entiers `n`$\in \mathbb{N}^*$ admettent 1 comme diviseur, donc on démarre notre compteur à 1.

Puis on effectue pour tous les entiers de 2 à `n` le test de divisibilité.
Afin de parcourir tous les entiers de 2 à `n` on utilise une boucle `for`, la variable choisie ici est `i`.

`range(2,n+1)` correspond aux entiers compris entre 2 (au sens large) et `n+1` (au sens strict). Autrement dit :

`range(2,n+1)`$=\{i \in \mathbb{N} \mid 2 \leq i < n+1\} = \{i \in \mathbb{N} \mid 2 \leq i \leq n\}$

Ce qui explique la présence de `n+1` dans l'instruction `range`.

A l'intérieur de la boucle, on effectue le test de divisibilité de `i` par `n` en réutilisant la fonction `divise` précédente. On sait que cette fonction renvoie le booléen `True` ou `False`.

Juste après l'écriture de l'instruction `if` on doit écrire un booléen. Etant donné que la fonction `divise` renvoie un booléen on n'a pas eu besoin d'écrire : `if divise(i,n)==True:`
Mais on pouvait le faire car `divise(i,n)==True` est aussi un booléen !

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0018

```
def nbdiviseur(n):
  k=1
  for i in range(2,n+1):
    if divise(i,n):
      k=k+1
  return k
```

## Fonction `listediviseurs`

3°) La liste des diviseurs de l'entier `n`$\in \mathbb{N}^*$ est initialisée avec la valeur 1 car 1 est un diviseur de `n`.

Puis on parcourt tous les entiers de 2 à `n` avec l'instruction :

`for i in range(2,n+1):`

A chaque fois que `i` divise `n` on ajoute `i` à la liste `liste` avec l'instruction `liste.append(i)`. Lorsque la boucle est achevée, `liste` est renvoyée.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0025

```
def listediviseur(n):
  liste=[1]
  for i in range(2,n+1):
    if divise(i,n):
      liste.append(i)
  return liste
```

## Fonction `listedivcommun`

4°) `la` et `lb` représentent respectivement la liste de diviseurs de `a` et `b`.

On initialise la liste `liste` contenant les diviseurs communs de `a` et `b` comme une liste vide.

`for i in la:` signifie que `i` va parcourir tous les éléments de la liste `la`. Donc `i` va prendre comme valeur successivement tous les diviseurs de `a`.

On teste très facilement avec Python la présence de `i` dans la liste `lb` des diviseurs de `b` en écrivant tout simplement `if i in lb`.

Si ce dernier test est positif alors `i` est un diviseur commun à `a` et `b`, on l'ajoute à la liste `liste` des diviseurs communs en écrivant :

`liste.append(liste)` . A la fin de la fonction on renvoie `liste`.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0032

```
def listedivcommun(a,b):
  la=listediviseur(a)
  lb=listediviseur(b)
  liste=[]
  for i in la:
    if i in lb:
      liste.append(i)
  return liste
```

*Thème: Arithmétique*

TI-83 Premium CE

# Diviseurs, nombres premiers et pgcd.

## Fonction `pgcd`

5°) Pour trouver le pgcd il suffit d'utiliser les fonctions que nous avons créées précédemment. Il correspond au plus grand diviseur commun, le script de la fonction tient donc en 4 lignes !

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0041

```
def pgcd(a,b):
  liste=listedivcommun(a,b)
  d=max(liste)
  return d
```

## Fonction `ppcm`

6°) En utilisant la propriété $\forall (a,b) \in \mathbb{N}^{*2}\ pgcd(a,b) \times ppcm(a,b) = ab$

On peut définir le ppcm comme étant égal à $\frac{ab}{pgcd(a,b)}$.

On n'oubliera pas d'écrire // pour que le résultat de la division soit un entier.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0046

```
def ppcm(a,b):
  d=pgcd(a,b)
  m=a*b//d
  return m
```

## Fonction `premier`

7°) On peut aborder cette fonction de façon très simple à l'aide des fonctions précédentes. En effet $n \in \mathbb{N}^*$ est premier lorsqu'il admet exactement 2 diviseurs (1 et lui-même). On peut donc compter le nombre de diviseurs de $n$ et tester s'il y en a 2 :

On peut aussi s'affranchir des fonctions précédentes et utiliser une propriété classique :

$n \in \mathbb{N}, n \geq 2$ est premier lorsqu'il n'est divisible par aucun entier compris entre 2 et $\sqrt{n}$.

On prend la précaution de tester si $n$ vaut 1 et de renvoyer `False` si c'est le cas, car notre propriété est valable pour $n \geq 2$, il faut donc traiter le cas $n = 1$ à part.

`m` va représenter la partie entière de $\sqrt{n}$. La fonction partie entière est `floor`, qui se trouve dans la bibliothèque `math`, il faudra bien penser à la lire avant en écrivant : `from math import *`.

Puis on écrit une boucle `for` avec une variable `i` qui prendra toutes les valeurs entières entre 2 et `m` (ne pas oublier d'écrire `range(2,m+1)`). A chaque tour de boucle on teste si `i` divise `n` et lorsque c'est le cas, on renvoie `False`.

Finalement si la boucle se termine c'est que l'instruction `return False` ne s'est jamais exécutée (l'instruction `return` arrête l'exécution de la fonction et donc de la boucle) cela signifie qu'aucun des nombres testés n'est un diviseur de `n` donc `n` est premier, il faut donc renvoyer `True`.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0051

```
def premier(n):
  k=nbdiviseur(n)
  if k==2:
    return True
  else:
    return False
```

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0058

```
from math import *
def premier(n):
  if n==1:
    return False
  else:
    m=floor(sqrt(n))
    for i in range(2,m+1):
      if divise(i,n):
        return False
    return True
```

# Diviseurs, nombres premiers et pgcd.

## Fonction `npremier`

8°) `npremier(n)` doit renvoyer le n-ième nombre premier. On va introduire deux variables `k` et `i` : `k` va dénombrer le nombre de nombres premiers trouvés et `i` est un entier qui est initialisé à 2 et qui va augmenter de 1 en 1 afin de parcourir tous les entiers jusqu'à en avoir rencontré `n` premiers. Dans ce cas l'algorithme s'arrête (puisque `k` et `n` sont égaux).

On remarque qu'on renvoie `i-1` et non pas `i`, en effet, pour `k=n-1` la boucle continue de s'exécuter et lorsque `i` est premier alors la valeur de `k` passe à `n` (donc le résultat à renvoyer est `i`), mais `i` est incrémenté de 1… C'est donc bien `i-1` qu'il faut renvoyer.

On peut aussi incrémenter `i` de 2 en 2 pour accélérer la boucle. Il faudra pour cela modifier le début du script de la fonction en initialisant `k` et `i` respectivement à 1 et 3. Il faudra aussi envisager le cas `n=1` à part.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0072

```
def npremier(n):
  k=0
  i=2
  while k<n:
    if premier(i):
      k=k+1
    i=i+1
  return i-1
```

## Fonction `listepremier`

9°) Pour obtenir la liste des `n` premiers nombres premiers, on commence par initialiser la liste `liste` qui va les contenir en écrivant `liste=[]`

Puis dans une boucle `for` la variable `i` va prendre toutes les valeurs entières de 1 à `n`. A chaque tour de boucle on ajoute à la liste `liste` le nombre premier n°`i` grâce à la fonction `npremier` précédente.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0081

```
def listepremier(n):
  liste=[]
  for i in range(1,n+1):
    liste.append(npremier(i))
  return liste
```

## Fonction `premiersup`

10°) Pour trouver le premier nombre premier supérieur strict à `n`, commençons par définir une variable `k` initialisée à `n+1` (puisqu'on cherche un nombre premier supérieur strict à `n`) et tant que `k` n'est pas premier on l'incrémente. La boucle va s'arrêter au premier nombre premier trouvé.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0087

```
def premiersup(n):
  k=n+1
  while premier(k)==False:
    k=k+1
  return k
```