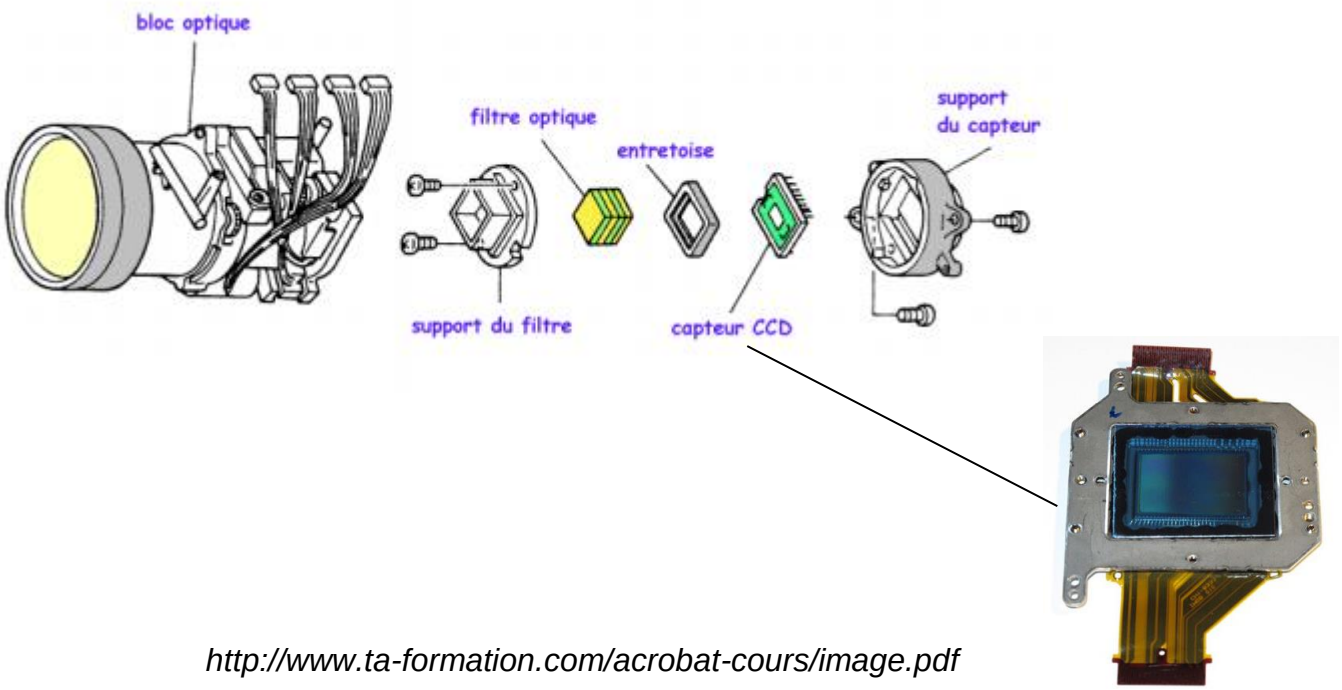
COULEURS & NUMERISATION, REALISTION D'UN « APPAREIL PHOTO » DE 4 PIXELS EN NIVEAU DE GRIS.

I. Principe appareil photo numérique :

Le schéma général d'un appareil photo numérique est donné ci-dessous :



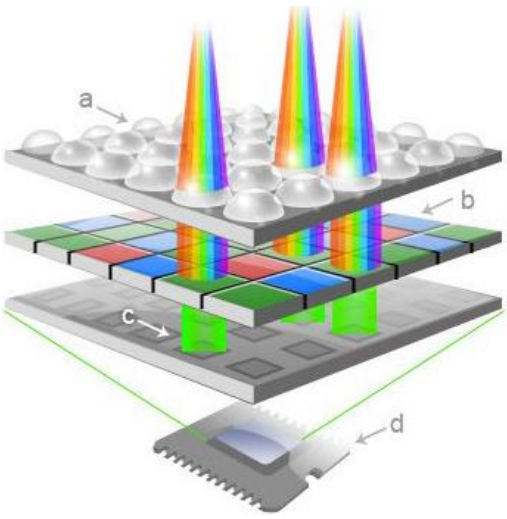
http://www.ta-formation.com/acrobat-cours/image.pdf

Les parties Filtre optique, capteur CCD sont détaillées dans le schéma ci-dessous :

Extrait de https://www.focus-numerique.com/reportage/dossiers/photo-en-noir-blanc-et-si-nous-enlevions-le-filtre-de-bayer-893.html

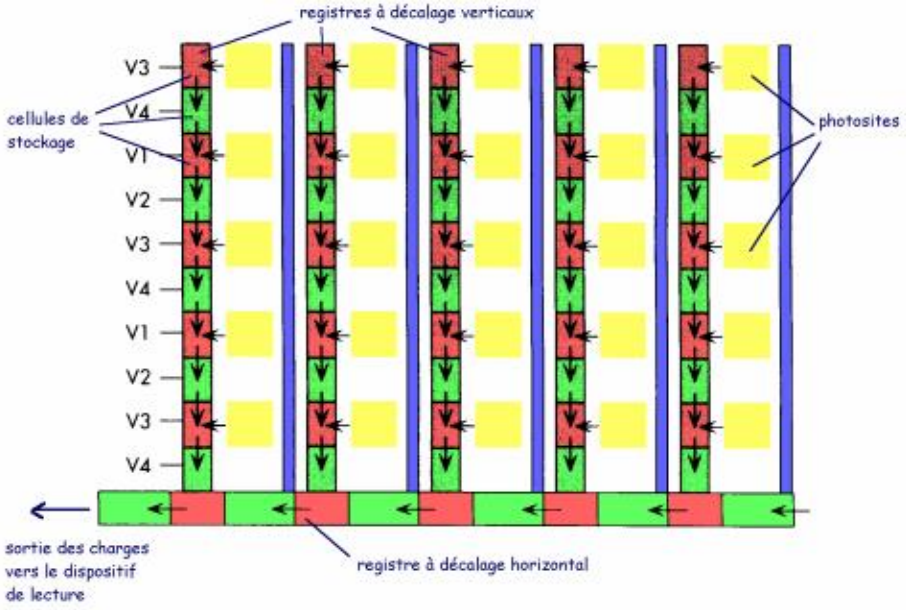
Les faisceaux lumineux sont « collimatés» par des microlentilles (a),

- Puis chaque faisceau est filtré, selon trois couleurs (Rouge Vert Bleu) par un filtre de Bayer (b). Cet ensemble de trois couleurs **forment un pixel**.
- Chaque couleur est alors quantifiée numériquement par une cellule photoélectrique ou photocite (c)
- Puis les données issues de chaque (d) cellule transitent par multiplexage comme indiqué ci-dessous :



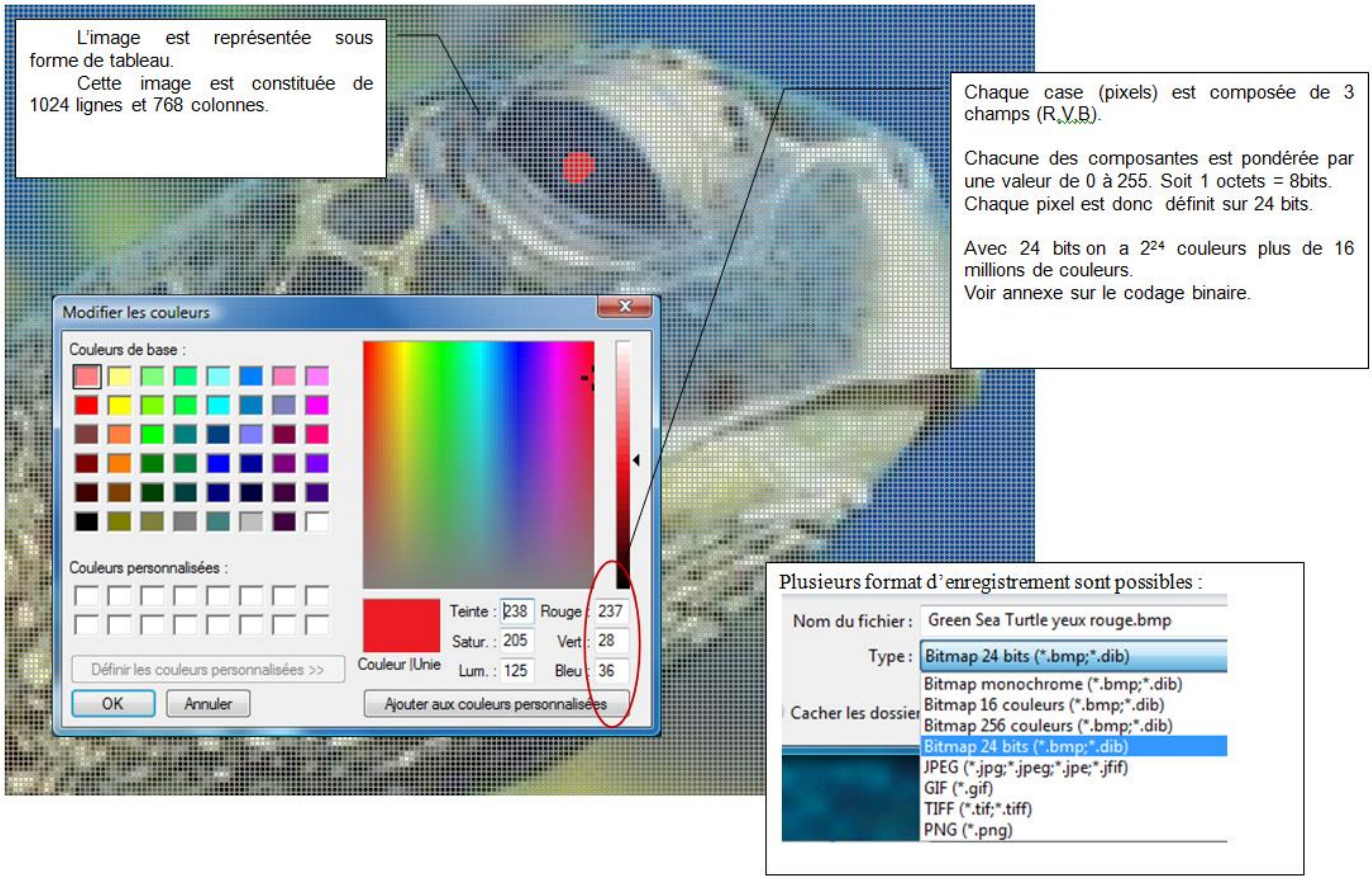
http://www.ta-formation.com/acrobat-cours/image.pdf

Le transfert des données des cellule est assuré par système de multiplexage présenté ci-dessous.



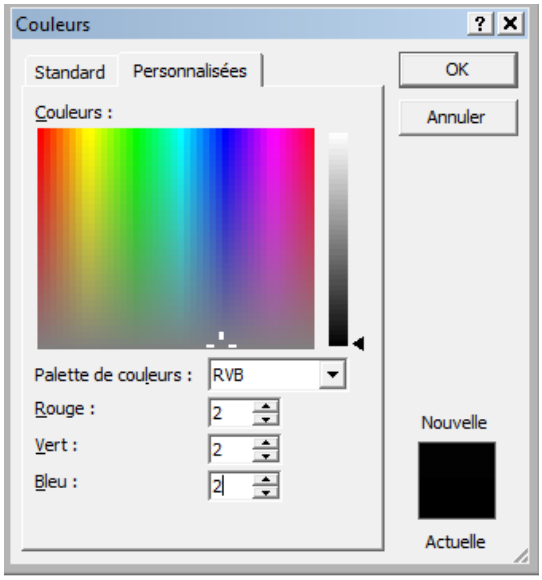
II. Couleur d'un pixel :

1. Pixel de couleurs

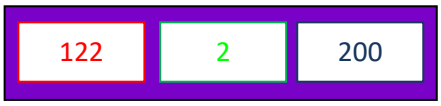


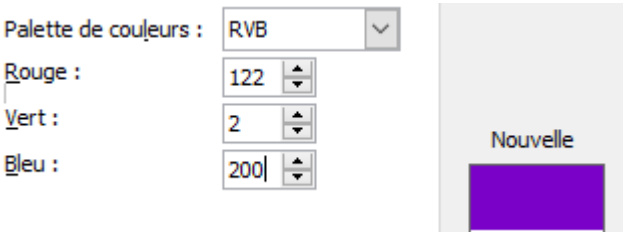
2. Pixel en niveau de gris

Q.1 *A partir de la fenêtre modification de couleur (sous open office par exemple) expliquer le rôle les paramètres R, V, B. Comment obtenir une image en niveaux de gris.*



Dans un ordinateur les trois canaux d'un pixel sont placés dans une seule case mémoire de 24 bits (3*8) comme le montre le schéma suivant :





Le nombre, sur 24 bits qui représente les 3 octets, sera donc dans cet exemple égal à $122*256^2 + 2*256 + 200$.

On peut définir une relation mathématique pour coder les couleurs sur 24 bits:

$N = Rouge * 256^2 + Vert *256 + Bleu$

Pour vérifier la relation précédente on se propose d'étudier le programme ci-dessous écrit en python, qui utilise tkinter.

Editor - C:\Users\Marc\Documents\Science numérique en Physique\un_pixel\frame de couleur.py

frame de couleur.py

```
1  from tkinter import *
2
3  fenetre = Tk()
4
5  fenetre['bg']='white'
6
7  # frame 1
8  Frame1 = Frame(fenetre, borderwidth=2, relief=GROOVE)
9  Frame1.pack(side=LEFT, padx=30, pady=30)
10
11 # frame 2
12 Frame2 = Frame(fenetre, borderwidth=2, relief=GROOVE)
13 Frame2.pack(side=LEFT, padx=10, pady=10)
14
15 # frame 3 dans frame 2
16 Rouge= 0
17 Vert =255
18 Bleu = 0
19 couleurEnDeci = Rouge*256**2+Vert*256+Bleu
20 couleur = '#%6.6x' %couleurEnDeci
21 Frame3 = Frame(Frame1, bg=couleur, borderwidth=2, relief=GROOVE)
22 Frame3.pack(side=RIGHT, padx=5, pady=5)
23
24 # Ajout de labels
25 Label(Frame1, text="Frame 1").pack(padx=10, pady=10)
26 Label(Frame2, text="Frame 2").pack(padx=10, pady=10)
27 Label(Frame3, text="Frame 3",bg="white").pack(padx=10, pady=10)
28
29 fenetre.mainloop()
```

Q-2 Dans quelle ligne peut-on vérifier le codage de la relation mathématique définie plus haut ?

Q-3 A partir de l'annexe 1, expliquer comment sont codées les couleurs sous python Dans quelle ligne du programme trouve-t-on cette conversion ?

Q4 Combien de caractères faut-il, pour coder en hexadécimal, un nombre sur 24 bits ? Vérifier la cohérence de votre réponse avec la ligne 20 du code.

Remarque : « option .6 » force la variable à avoir 6 caractères même si le caractère de poids fort est « 0 ».

III. But du TP : Réaliser « un appareil photo numérique 4 pixels en niveau de gris ».

A partir de quatre capteurs de luminosité, on désire étudier le principe simplifié de la numérisation de 4 pixels en niveau de gris.
La partie optoélectronique analogique sera réalisée avec un TIL78,
La numérisation sera faite en utilisant le CAN d'un Arduino (CAN 10 Bits),
L'affichage sera fait sur un PC, la transmission de l'information entre l'Arduino et le PC se fera par USB.

Dans les parties 1 et 2 on se limitera pour l'instant à 1 pixel.

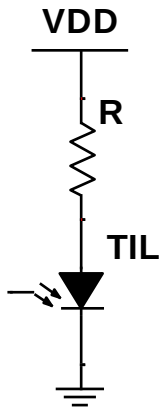
1. Etude de la partie optoélectronique :

Q.5 *A partir de la documentation du module Arduino, déterminer la plage de mesure du CAN. Ainsi que le nombre de bits du CAN.*

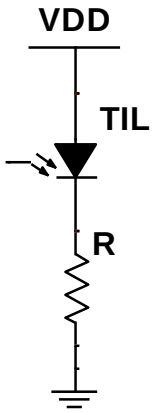
On peut considérer que le composant TIL 78 se comporte comme une résistance variant de R=0 à R= infini suivant l'éclairement.

Q.6 *Expliquer dans quel cas R= 0 et R= infini (l'explication sera donnée à partir des notions d'énergie d'un photon, de niveau d'énergie atomique, de bande de valence et de conduction).*

La documentation technique du TIL 78 propose les deux montages suivants :



montage a



montage b

Q.7 *Choisir le bon montage, pour que lorsque le phototransistor est éclairé au maximum, la tension en sortie du montage, corresponde à la pleine échelle du CAN.*

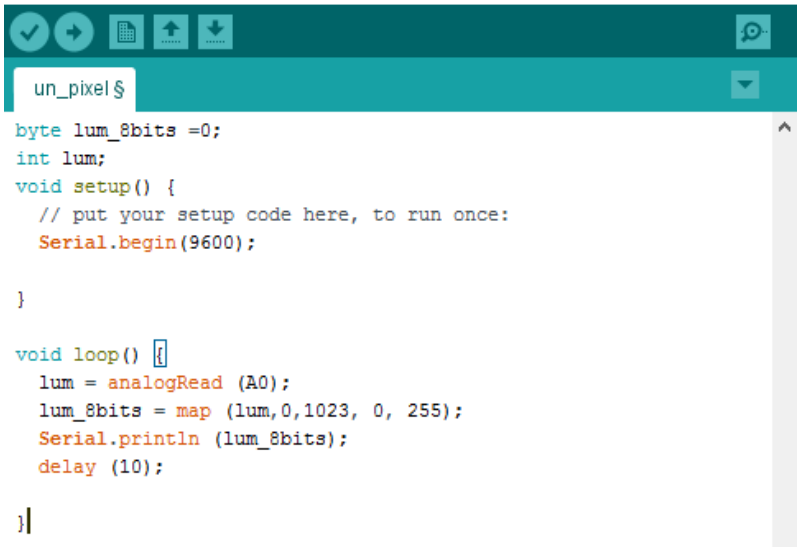
Q-8 *Choisir la valeur à donner à la tension VDD et à la résistance sachant que le courant max que supporte le TIL78 est de 1mA.*

2. Mise en œuvre

Q-9 *Réaliser le montage et programmer Arduino pour qu'une valeur représentative de l'éclairement soit renvoyeé au PC via l'USB.*

Q-10 *Sur combien de bits est codée l'information numérisée par Arduino? Sur combien de bit doit être codée la couleur sur le PC ? y a-t-il un problème de compatibilité ?*

Pour remédier à ce problème on se propose d'utiliser le code suivant :



Q-11 *Identifier la partie du code qui apporte une solution au problème.*
Expliquer ce qui est réalisé en s'aidant de l'annexe 2

Q-12 *Réaliser un programme sur le PC qui récupère les données sur le port USB et affiche un carré (représentant un pixel) dont le niveau de gris dépend de l'éclairement.*
Ce programme peut être réalisé :

- en Python en utilisant le module tkinter pour l'interface,
- sous Labview, avec tout autre langage de programmation.

3. Mise en œuvre du système complet :

Q-13 Modifier l'ensemble du système pour afficher 4 pixels.

Les quatre valeurs seront transmisses dans un même « mot», elles seront séparées par une virgule, et la fin du mot sera indiqué par le caractère « \r » (pour cela la dernière commande sera d'écriture du programme d'Arduino sera Serial.println()).

Annexe 1 : TKINTER

http://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel

Hello world

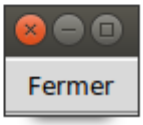
Voici le code de votre premier hello world

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from tkinter import *

fenetre = Tk()

label = Label(fenetre, text="Hello World")
label.pack()

fenetre.mainloop()
```

Les boutons

Les boutons permettent de proposer une action à l'utilisateur. fermer la fenêtre.

```
# bouton de sortie
bouton=Button(fenetre, text="Fermer", command=fenetre.quit)
bouton.pack()
```



Frames

Les frames (cadres) sont des conteneurs qui permettent de séparer des éléments.

```
fenetre['bg']='white'

# frame 1
Frame1 = Frame(fenetre, borderwidth=2, relief=GROOVE)
Frame1.pack(side=LEFT, padx=30, pady=30)

# frame 2
Frame2 = Frame(fenetre, borderwidth=2, relief=GROOVE)
Frame2.pack(side=LEFT, padx=10, pady=10)

# frame 3 dans frame 2
Frame3 = Frame(Frame2, bg="white", borderwidth=2, relief=GROOVE)
Frame3.pack(side=RIGHT, padx=5, pady=5)

# Ajout de labels
Label(Frame1, text="Frame 1").pack(padx=10, pady=10)
Label(Frame2, text="Frame 2").pack(padx=10, pady=10)
Label(Frame3, text="Frame 3",bg="white").pack(padx=10, pady=10)
```

Les options de couleurs

Il est possible d'indiquer une valeur de couleur par son nom en anglais: "white", "black", "red", "yellow", etc. ou par son code hexadécimale: #000000, #00FFFF, etc.

```
background (ou bg)   : couleur de fond du widget.
foreground (ou fg)   : couleur de premier plan du widget.
activebackground     : couleur de fond du widget lorsque celui-ci est actif.
activeForeground     : couleur de premier plan du widget lorsque le widget est actif.
disabledForeground   : couleur de premier plan du widget lorsque le widget est désactivé.
highlightbackground  : Couleur de fond de la région de surbrillance lorsque le widget a le focus.
highlightcolor       : couleur de premier plan de la région en surbrillance lorsque le widget a le focus.
selectbackground     : Couleur de fond pour les éléments sélectionnés.
selectforeground     : couleur de premier plan pour les éléments sélectionnés.
```

Annexe 2 : Defining Data Types

The Arduino environment is really just C++ with library support and built-in assumptions about the target environment to simplify the coding process. C++ defines a number of different data types; here we'll talk only about those used in Arduino with an emphasis on traps awaiting the unwary Arduino programmer.

Below is a list of the data types commonly seen in Arduino, with the memory size of each in parentheses after the type name. Note: **signed** variables allow both positive and negative numbers, while **unsigned** variables allow only positive values.

- **boolean** (8 bit) - simple logical true/false
- **byte** (8 bit) - unsigned number from 0-255
- **char** (8 bit) - signed number from -128 to 127. The compiler will attempt to interpret this data type as a character in some circumstances, which may yield unexpected results
- **unsigned char** (8 bit) - same as 'byte'; if this is what you're after, you should use 'byte' instead, for reasons of clarity
- **word** (16 bit) - unsigned number from 0-65535
- **unsigned int** (16 bit)- the same as 'word'. Use 'word' instead for clarity and brevity
- **int** (16 bit) - signed number from -32768 to 32767. This is most commonly what you see used for general purpose variables in Arduino example code provided with the IDE
- **unsigned long** (32 bit) - unsigned number from 0-4,294,967,295. The most common usage of this is to store the result of the `millis()` function, which returns the number of milliseconds the current code has been running
- **long** (32 bit) - signed number from -2,147,483,648 to 2,147,483,647
- **float** (32 bit) - signed number from -3.4028235E38 to 3.4028235E38. Floating point on the Arduino is not native; the compiler has to jump through hoops to make it work. If you can avoid it, you should. We'll touch on this later.

http://apprendre

This tutorial will **NOT** cover arrays, pointers, or strings; those are more specialized datatypes with more involved concepts that will be covered elsewhere.

http://enacit1.epfl.ch/introduction-python/index.html#standard-library

map(value, fromLow, fromHigh, toLow, toHigh)

DESCRIPTION

Ré-étalonne un nombre d'une fourchette de valeur vers une autre fourchette. Ainsi, une valeur basse source sera étalonnée en une valeur basse de destination, une valeur haute source sera étalonnée en une valeur haute de destination, une valeur entre les deux valeurs source sera étalonnée en une valeur entre les deux valeurs destinations, en respectant la proportionnalité. Cette fonction est très utile pour effectuer des changements d'échelle automatiques.

Cette fonction ne contraint pas les valeurs à rester dans les limites indiquées, car les valeurs en dehors de la fourchette sont parfois attendues et utiles. L'instruction constrain() doit être utilisée également avant ou après cette fonction, si les limites de la fourchette utilisée doivent être respectées.

Noter que la limite basse de chaque fourchette peut être supérieure ou inférieure à la limite haute, dès lors l'instruction map() peut être utilisée pour inverser l'ordre des valeurs, par exemple :

```
y = map(x, 1, 50, 50, 1); // y évolue en sens inverse de x (càd si x = 1, y=50 et inversement)
```

[Obtenir le Code]

Cette instruction supporte également des valeurs négatives, tel que dans cet exemple :

```
y = map(x, 1, 50, 50, -100);
```

[Obtenir le Code]

Cette utilisation est aussi valide et fonctionne normalement.

L'instruction map() utilise des valeurs entières qui ne peuvent fournir les décimales, alors que les calculs le devraient. La partie décimale est tronquée, et les valeurs ne sont pas arrondies ou moyennées.

SYNTAXE

```
map (valeur, limite_basse_source, limite_haute_source, limite_basse_destination, limite_haute_destination)
```

[Obtenir le Code]

ANNEXE 3 :

- **méthode split**

L'extrait du site http://www.esi.umontreal.ca/~mousseau/phy1234/notes/notes_15.html donne un exemple de l'utilisation de la méthode.

```
>>> s = 'ceci;est;un;mouton'
   >>> s.split()
   ['ceci;est;un;mouton']
   >>> mots = s.split(';')
   >>> print mots
   ['ceci', 'est', 'un', 'mouton']
```

- **Conversion d'une chaine au format tableur en un tableau de nombre**

```
 8 hello = "12,44,255"
 9 liste=hello.split(',')
10 #chaque élément de la liste est une chaîne de caractères
11 # on ne peut pas faire de calcul sur une chaîne de caractères
12 print (liste)
13 # "int" dans la ligne dessous convertie la chaîne en nombre
14 S= 2*int(liste[0])
15 print (S)
```