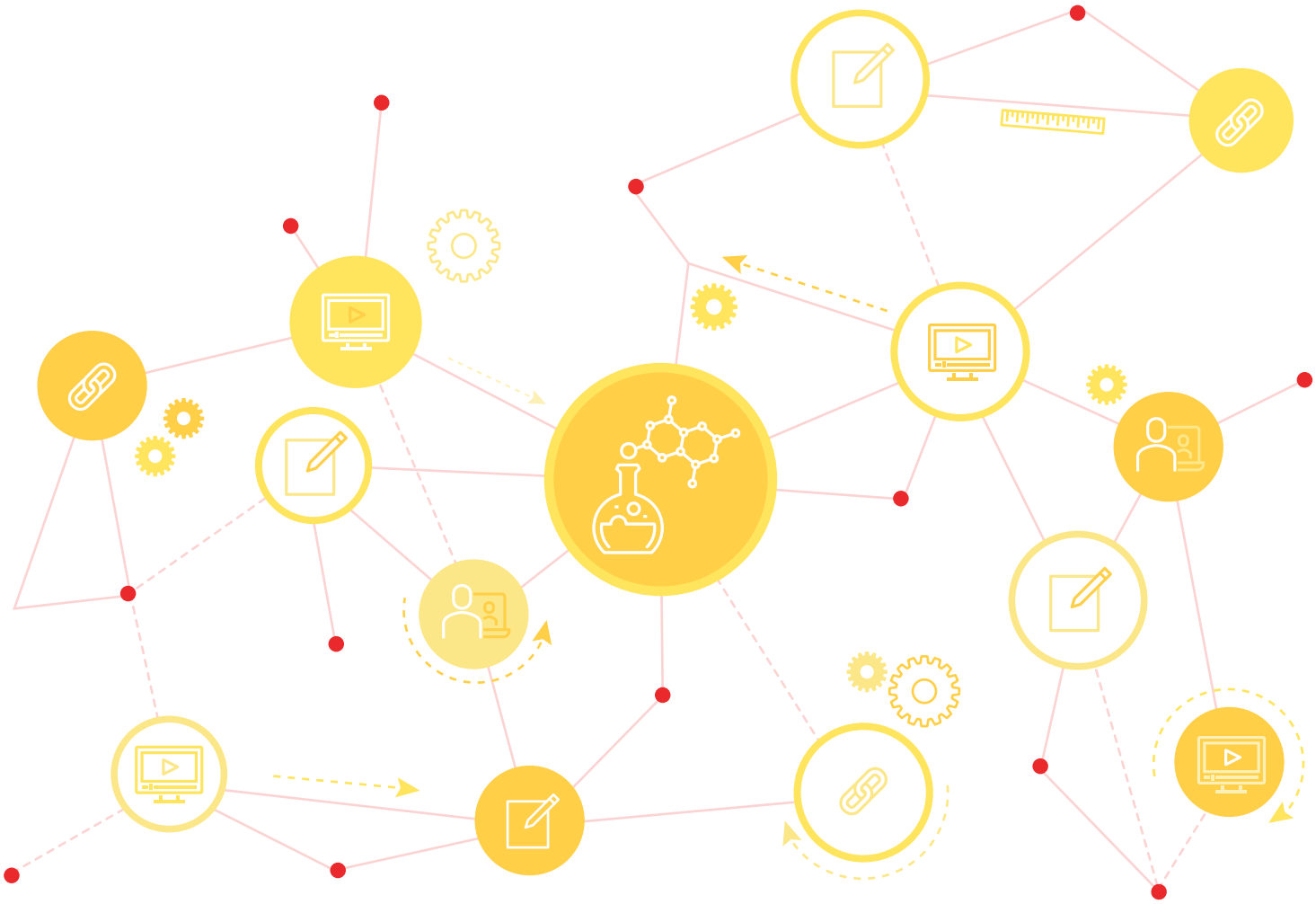
T³ FRANCE

Teachers Teaching with Technology™



Stéphane Bonnaud et Eric Tixidor

# Livret d'activités en physique-chimie
# Lycée général

www.t3france.fr



python™

# Sommaire

## Référentiel, compétences

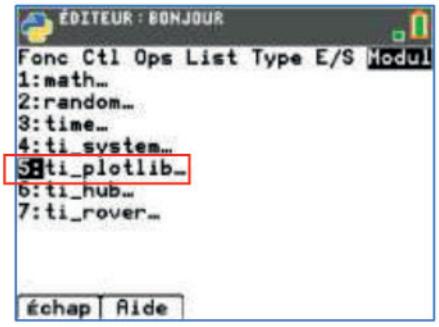
- **Capacité numérique :** représenter un nuage de points associé à la caractéristique d'un dipôle et modéliser la caractéristique de ce dipôle à l'aide d'un langage de programmation.

## Commentaires de l'auteur

- Grâce au module **ti_plotlib** et à la fonction **scatter**, développés spécifiquement par Texas Instruments, il est possible de représenter un nuage de points sur la calculatrice. Notons que la fonction **plot**, utilisée à la place de **scatter**, aurait relié les points entre eux. *Scatter* signifie dispersion.
- Afin de représenter graphiquement des données numériques, nous pouvons au choix :
  - ✓ Exploiter les données stockées dans deux listes Python, déclarées directement dans le programme Python. Pour cela, aller à l'Étape 2.
  - ✓ Exploiter les données stockées dans deux listes du menu STATS (**$L_1$** et **$L_2$** par exemple). Pour cela, aller directement à l'Étape 3.
- Le script écrit en langage Python affiche le nuage de points correspondant aux données numériques.
- Il sera ensuite possible soit de rechercher manuellement un modèle mathématique (par tâtonnement), soit d'effectuer une régression linéaire.

## Matériel

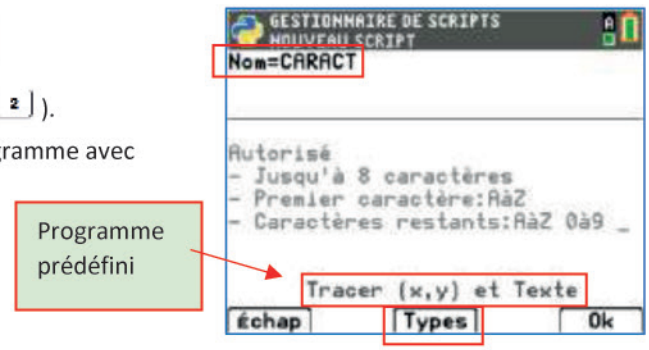
- Calculatrice TI-83 Premium CE Edition Python.
- Module Python **ti_plot_lib** préalablement installé. Sur la figure ci-contre, il apparaît dans le menu Modul en 5ème position.
- Coordonnées successives $(U, I)$ du dipôle étudié. En guise d'exemple, nous allons utiliser les mesures suivantes qui illustrent la loi d'Ohm pour un conducteur ohmique :




| $U(V)$ | 0 | 0.67 | 1.27 | 2.03 | 2.73 | 3.49 | 4.21 | 4.94 |
|---|---|---|---|---|---|---|---|---|
| $I(mA)$ | 0 | 3.0 | 5.7 | 9.1 | 12.3 | 15.7 | 19.0 | 22.3 |

## Étape 1 : Préparer le script Python

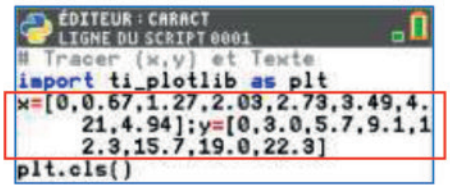
- Aller dans l'application **Python App**, touche [prgm] puis [2] ).
- Créer un nouveau script avec [zoom], choisir le type de programme avec **Types** ( touche [zoom] puis [4] ).
- Donner un nom au script, ici **CARACT**.
- Valider par **Ok**, touche [graphe].

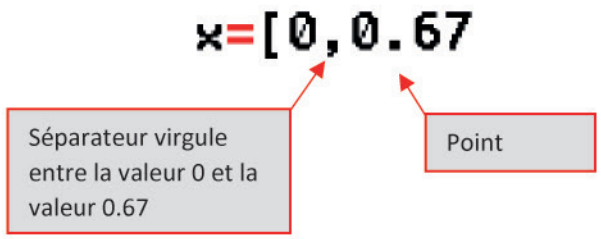



Programme prédéfini

## Étape 2 : Saisir les deux listes Python

- Compléter les deux listes Python nommées **x** et **y** avec les données à représenter. Le point-virgule qui sépare les deux listes **x** et **y** peut être remplacé par un retour à la ligne.
- Attention à ne pas confondre la *virgule* (qui joue le rôle de séparateur en Python) et le *point*, qui est le point décimal.



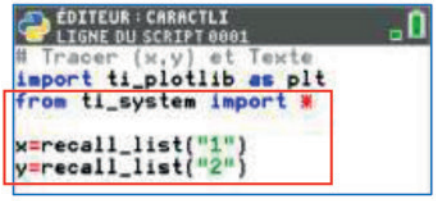




## Étape 3 : Saisir les données à représenter

Nous allons utiliser les deux listes $L_1$ et $L_2$ du menu **STATS.**

- Pour utiliser les listes $L_1$ et $L_2$, il faut légèrement modifier le script **CARACT**. Pour cela,
  1. Importer le module **ti_system** en début de script.
  2. Remplacer **x=[…] ; y=[…]** par :

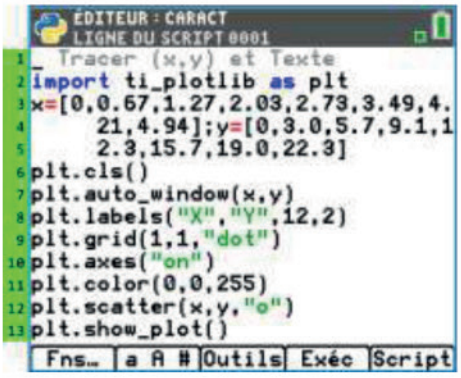
     ```
     x=recall_list("1")
     y=recall_list("2").
     ```




Ainsi, le contenu des listes $L_1$ et $L_2$ du menu **STATS** sera stocké dans les variables **x** et **y** respectivement. Attention, ces deux listes doivent avoir la même taille.

## Étape 4 : Détails du code

Vous trouverez ci-contre le code complet permettant l'affichage du nuage de points, selon la méthode décrite à l'étape 2. Télécharger le script **CARACT** à l'adresse : https://education.ti.com/fr/physique-chimie.




- Ligne 2 : Importation du module **ti_plot_lib**.
- Ligne 3 : Création des deux listes **x** et **y**, comme vu à l'Étape 2.
- Ligne 6 : Efface l'écran.
- Ligne 7 : Paramètre la fenêtre d'affichage.
- Ligne 8 : Affiche le nom des deux axes.
- Ligne 9 : Affiche un quadrillage de côté 1 unité avec le style "dot".
- Ligne 10 : Affiche les axes du graphique.
- Ligne 11 : Paramètre la couleur du nuage de points en (R, V, B), ici en bleu (0,0,255).
- Ligne 12 : Trace le nuage de point à l'aide des valeurs des deux listes **x** et **y**, avec la marque "o".
- Ligne 13 : Affiche le graphique à l'écran de manière continue, jusqu'à ce que la touche **ON** soit pressée pour revenir au Shell.

# Étape 5 : Modéliser la caractéristique du dipôle par tâtonnement

Nous allons à présent modéliser la caractéristique du dipôle ohmique par tâtonnement.

- Nous allons ici faire une recherche manuelle de modèle mathématique. Il s'agit de superposer sur le nuage de points précédemment tracé la représentation graphique d'une droite affine d'équation $y = a \cdot x + b$.
- En faisant varier les valeurs des coefficients $a$ et $b$, il est possible d'ajuster manuellement la droite pour qu'elle passe au plus près des points expérimentaux. Nous pouvons ainsi déterminer l'équation de la droite modèle.

Pour cela, nous allons :

1. Écrire une fonction Python **f**, qui a pour argument une liste d'abscisses appelée **L** et qui renverra la liste des images des abscisses par la fonction affine $y = a \cdot x + b$ (Lignes 7 à 13).
2. Superposer la droite modèle sur le même graphique.

Télécharger le script **CARACTB** qui propose de modéliser manuellement la caractéristique du dipôle, à l'adresse : https://education.ti.com/fr/physique-chimie.

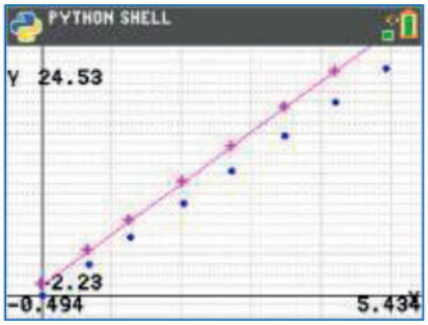
- Modifier par tâtonnement les valeurs de **a** et de **b**, directement dans le script (Lignes 8 et 9).
- Exécuter le script à nouveau pour obtenir un nouveau tracé.


**Remarques :**

- La ligne 21 paramètre la couleur du graphique de la ligne 22, ici en bleu (0,0,255) selon la norme (R,V,B).
- La ligne 24 paramètre la couleur du graphique de la ligne 25, ici en magenta (255,0,255).

Ci-contre, un exemple de sortie graphique :

- ✓ En bleu le nuage de points.
- ✓ En magenta, la droite affine modèle qu'il convient d'ajuster manuellement en faisant varier les valeurs de **a** et de **b** dans le script. Sur le schéma ci-contre, les valeurs de **a** et de **b** sont trop élevées, il faudrait les diminuer.

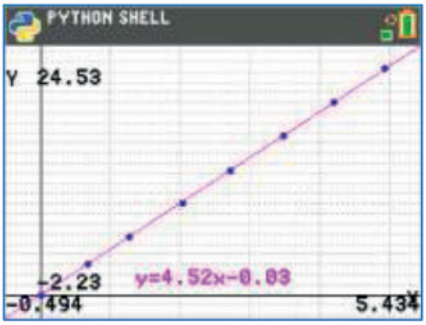


## Étape 6 : Modéliser la caractéristique par régression linéaire

Si on le souhaite, il est possible de tracer la droite modèle grâce à la fonction **lin_reg** du module **ti_plot_lib** (*linear regression*).

- Le script est modifié de la façon suivante (voir ci-contre) :
  - ✓ La ligne 17 effectue la régression linéaire et affiche la droite de régression linéaire sur le graphique.
  - ✓ Les deux arguments "center",**11** permettent l'affichage de l'équation de la droite modèle, centrée, et en bas du graphique.
- Télécharger le script **CARACREG** à l'adresse : https://education.ti.com/fr/physique-chimie.


- La sortie graphique est représentée ci-contre. L'équation affichée en magenta est l'équation de la droite modèle.




## Étape 7 : Cas où les données sont stockées dans $L_1$ et $L_2$

Nous avons vu à l'étape 3 comment utiliser dans Python le contenu des listes $L_1$ et $L_2$ du menu STATS. Pour effectuer une régression linéaire, nous devons :

1. Importer les listes $L_1$ et $L_2$ comme à l'étape 3,
2. Effectuer normalement la régression linéaire comme à l'étape 6.

## Référentiel, compétences

- **Capacité numérique :** représenter des vecteurs vitesse d'un système modélisé par un point lors d'un mouvement, à l'aide d'un langage de programmation.

## Commentaires de l'auteur

- Grâce au module ***ce_ quiver***, développé spécifiquement par Texas Instruments, il est possible de représenter des vecteurs vitesse à différents instants d'un mouvement.
- Le programme écrit en langage Python affiche point par point les positions successives du système étudié ainsi que les vecteurs vitesses associés.

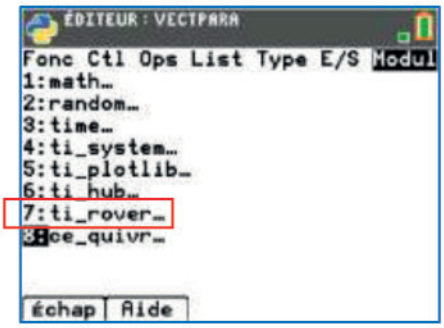



Fig. 1

## Matériel

- Calculatrice TI-83 premium CE Edition Python.
- **Module** Python **ce_ quiver** préalablement installé. Sur la figure 1, il apparaît dans le menu **Modul** en 8ème position.
- Coordonnées successives $(x, y)$ du point matériel à étudier (7 à 15 points, pour une meilleure lisibilité sur l'écran de la calculatrice). Dans cette fiche, nous allons utiliser un ensemble de 7 points formant une trajectoire parabolique :

| $x(m)$ | 0 | 0.76 | 1.52 | 2.28 | 3.04 | 3.8 | 4.56 |
|---|---|---|---|---|---|---|---|
| $y(m)$ | 2.2 | 3.32 | 4.05 | 4.38 | 4.32 | 3.87 | 3.03 |

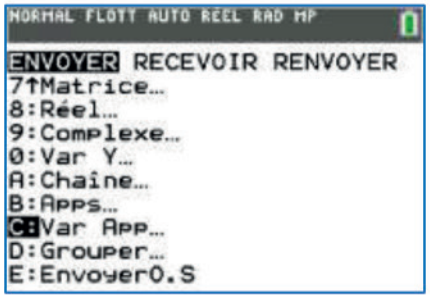



Fig. 2

## Prérequis

- Pour installer le module ***ce_quiver*** sur la calculatrice, on peut procéder de deux manières *:*
  - ✓ L'envoyer depuis un ordinateur,
  - ✓ L'envoyer depuis une autre calculatrice : le module se trouve dans le menu **ENVOYER** **C :** **Var App…** (fig.2)

## Étape 1 : Vérification des listes

- Vérifier que les deux listes $L_1$ et $L_2$ contiennent bien les coordonnées successives du système étudié (fig. 3). Ces valeurs peuvent être issues d'une expérience (acquises par l'intermédiaire du Microcontrôleur TI-Innovator™ Hub, par exemple) ou avoir été entrées manuellement.

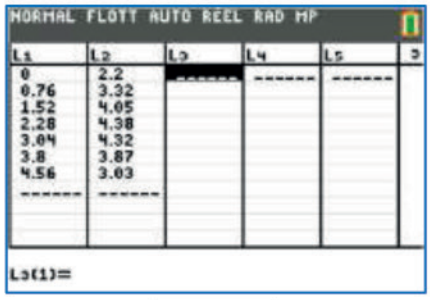



Fig. 3

## Étape 2 : Édition du script VECTPARA

- Depuis l'application ***Python App***, *Éditer* le fichier **VECTPARA**, préalablement transféré depuis l'ordinateur ou une autre calculatrice (fig. 4). On obtient l'écran de la figure 4.

  Télécharger le programme **VECTPARA** à l'adresse suivante :

  https://education.ti.com/fr/physique-chimie.
- Le programme compte 23 lignes. Les lignes 4 et 8 ont été insérées pour la lisibilité du code. La ligne 14 est un retour à la ligne indispensable.

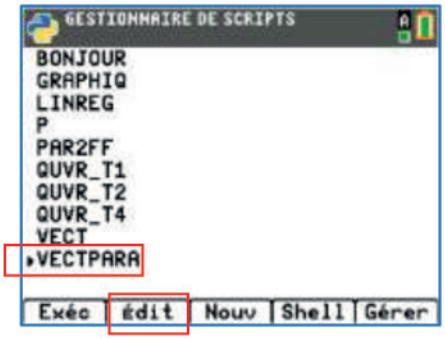



Fig. 4

## Étape 3 : Détails du code

- Lignes 5 et 6 : les listes **x1** et **y1** reçoivent par affectation les données des listes $L_1$ et $L_2$ du module STATS. Il est possible de changer les valeurs `"1"` et `"2"` si nécessaire.
- Ligne 7 : la variable **dt** reçoit la valeur de l'écart temporel entre deux positions successives du système étudié.
- Lignes 9 à 22 : la fonction **trace** permet le tracé des vecteurs, munie des deux arguments **x** et **y**.
- Ligne 16 : la boucle **for** permet de parcourir de manière itérative les listes **x** et **y**, **i** variant de **0** à **5**. Les vecteurs vitesse des six premiers points seront ainsi tracés pour des listes **x** et **y** de taille 7.
- Lignes 17 et 18 : calcul des coordonnées **vx** et **vy** du vecteur vitesse du point n°**i**.
- Lignes 19 et 20 : appel de la fonction **quiver** qui dessine le vecteur vitesse au point n°**i**. Le paramètre **0.1** est un facteur d'échelle qu'il est possible de faire varier pour rendre les vecteurs plus ou moins longs.
- Ligne 21 : tracé d'une grille de côté 1/5 et de style `"dot"`.
- Ligne 23 : appel à la fonction **trace** avec les arguments **x1** et **y1**.

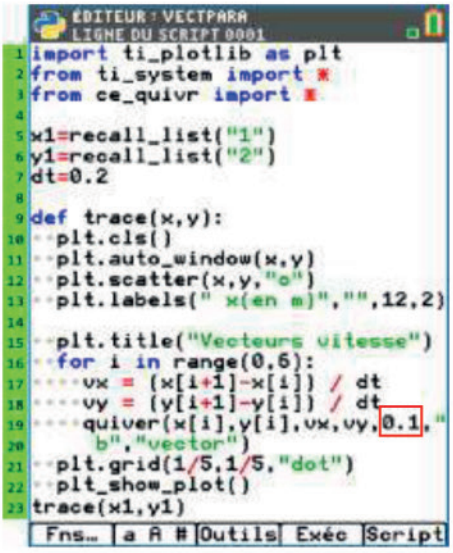



Fig. 5

## Étape 4 : Conclusion

- On observe que les vecteurs vitesse sont toujours tangents à la trajectoire et qu'ils sont orientés dans le sens de la trajectoire.
- Le vecteur vitesse est d'autant plus grand que la vitesse est grande.
- Il pourra être judicieux de faire varier les trajectoires (rectiligne, circulaire, parabolique) afin d'observer les vecteurs vitesses associés.
- Dans le cas d'un mouvement rapide, on pourra ne prendre par exemple qu'un point sur 2 ou un point sur 3.
- Un prolongement à cette fiche méthode existe pour le tracé des vecteurs accélération, en spécialité physique-chimie de la classe de terminale.

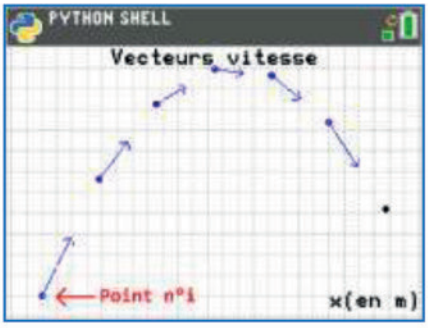



Fig. 6

# Fiche méthode

## Référentiel, compétences

- Utiliser un dispositif (smartphone, logiciel de traitement d'images, etc.) pour étudier l'évolution des énergies cinétique, potentielle et mécanique d'un système dans différentes situations : chute d'un corps, rebond sur un support, oscillations d'un pendule, etc.
- **Capacité numérique** : Utiliser un langage de programmation pour effectuer le bilan énergétique d'un système en mouvement.

## Commentaires de l'auteur

- Les logiciels de pointage utilisés en mécanique lors d'une chronophotographie proposent souvent d'afficher les évolutions temporelles de la vitesse du système étudié. Les énergie $Ec$, $Epp$ et $Em$ peuvent aussi être affichées en fonction du temps.
- Il est intéressant de réaliser cette même étude à l'aide d'un langage de programmation. En effet, le programme effectue étape par étape les différents traitements mathématiques. Entièrement paramétrable, le programme est évolutif et peut être modifié par l'élève selon ses besoins (en affichage ou en calcul).
- Seules les données numériques $x(t)$ et $y(t)$ sont nécessaires en entrée du programme. Les données brutes utilisées dans cette fiche en guise d'exemple sont extraites du lancer d'une balle de golf dans le champ de pesanteur. Un traitement de ces mêmes données numériques avec le logiciel *AVISTEP* est disponible sur YouTube.

## Matériel

- Calculatrice TI-83 Premium CE Edition Python.
- Coordonnées successives $(t, x, y)$ du point matériel à étudier (fig.1).

### Étape 1 : Vérification des listes

- Vérifier que les trois listes **L1**, **L2** et **L3** du menu **STATS** contiennent bien les données de temps (**L1**) et les coordonnées successives du système étudié (**L2** et **L3**).
- L'instruction **`t=recall_list("1")`** permet de stocker dans la variable **t** le contenu de la liste **L1** du menu **STATS**. Idem pour **x1** et **y1**.

```
ÉDITEUR : ENERGIE
LIGNE DU SCRIPT 0001
_mport ti_plotlib as plt
from ti_system import *
m=0.045
g=9.81

t=recall_list("1")
x1=recall_list("2")
y1=recall_list("3")
dt=t[1]-t[0]
Ec=[];Epp=[];Em=[]
Fns… a A # Outils Exéc Script
```

Fig. 2

| $t(s)$ | $x(m)$ | $y(m)$ |
|---|---|---|
| 0.000 | 0.010 | 0.055 |
| 0.040 | 0.084 | 0.180 |
| 0.080 | 0.161 | 0.293 |
| 0.120 | 0.235 | 0.394 |
| 0.160 | 0.314 | 0.481 |
| 0.200 | 0.393 | 0.554 |
| 0.240 | 0.474 | 0.612 |
| 0.280 | 0.555 | 0.657 |
| 0.320 | 0.636 | 0.685 |
| 0.360 | 0.720 | 0.697 |
| 0.400 | 0.802 | 0.692 |
| 0.440 | 0.884 | 0.671 |
| 0.480 | 0.968 | 0.633 |
| 0.520 | 1.052 | 0.576 |
| 0.560 | 1.136 | 0.502 |
| 0.600 | 1.219 | 0.411 |
| 0.640 | 1.303 | 0.300 |
| 0.680 | 1.385 | 0.170 |
| 0.720 | 1.477 | 0.026 |

Fig. 1

- Il est également possible de faire figurer ces valeurs numériques dans trois listes **t**, **x1** et **y1** directement dans le programme Python.

## Étape 2 : Édition du script *ENERGIE*

- Depuis l'application ***Python App***, *Éditer* le fichier **ENERGIE**, préalablement transféré depuis l'ordinateur ou une autre calculatrice. On obtient l'écran de la figure 3. Télécharger le programme **ENERGIE** à l'adresse https://education.ti.com/fr/physique-chimie.

  Le programme compte 40 lignes. Les lignes 5, 11, 23, 26, 30, 34 et 38 ont été insérées pour la lisibilité du code.


Fig. 3

## Étape 3 : Détails du code

- Ligne 2 : le module **ti_system** permet de récupérer le contenu des listes du menu **STATS**.
- Lignes 6, 7 et 8 : les listes **t**, **x1** et **y1** reçoivent par affectation les données des listes $L_1$, $L_2$ et $L_2$ du module STATS. Il est possible de changer les valeurs "1", "2" et "3"si nécessaire.
- Ligne 7 : la variable **dt** calcule la valeur de l'écart temporel entre deux données successives de la liste **t**.
- Lignes 12 à 39 : la fonction **trace,** munie des deux arguments **x** et **y**, permet le calcul et le tracé des énergies.
- Ligne 16 : la boucle **for** permet de parcourir de manière itérative les listes **t**, **x** et **y, i** variant de **0** à **len(t)-2** incluse.
- Lignes 17, 18 et 19 : calcul des coordonnées **vx** et **vy** du vecteur vitesse du point n°**i**, puis de sa norme **v**.
- Lignes 20, 21 et 22 : les énergies au point **i** sont calculées et ajoutées aux listes **Ec**, **Epp** et **Em**, qui ont été initialisés par des listes vides à la ligne 10.
- Ligne 24 : paramétrage d'une fenêtre de sortie, adaptée à la valeur des énergies de cette expérience.
- Ligne 25 : tracé d'une grille de côté 1/5 et de style "dot".
- Ligne 27 à 29 : mise en forme (couleur, style) tracé de la courbe **Ec**.
- Ligne 31 à 33 : mise en forme (couleur, style) et tracé de la courbe **Epp**.
- Ligne 35 à 37 : mise en forme (couleur, style) et tracé de la courbe **Em**.
- Ligne 40 : appel à la fonction **trace** avec les arguments **x1** et **y1**.

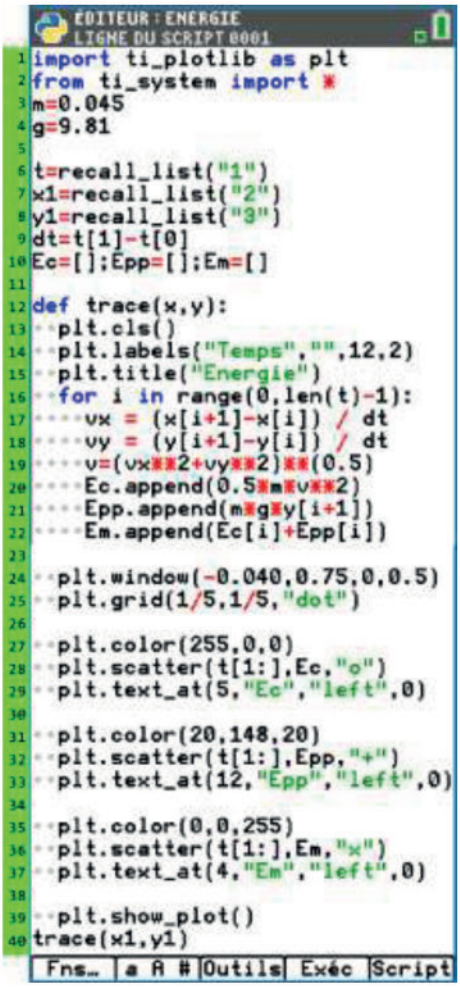



Fig. 4

## Étape 4 : Conclusion

- On observe qu'il y a interconversion de l'énergie au cours du temps. Au départ du mouvement, l'énergie est sous forme cinétique. Au sommet de la trajectoire, l'énergie est essentiellement est sous forme potentielle.
- Le mouvement d'une balle de golf dans le champ de pesanteur est un exemple de conservation de l'énergie mécanique. L'énergie mécanique totale du système, **Em**, est quasiment constante au cours du mouvement : il y a conservation de l'énergie mécanique. Nous verrons en spécialité physique-chimie en classe de première que ceci est dû au caractère conservatif des forces extérieures appliquées sur le système. Le poids $\vec{P}$ est une force conservative, car il dérive d'une énergie potentielle (ici **Epp**).

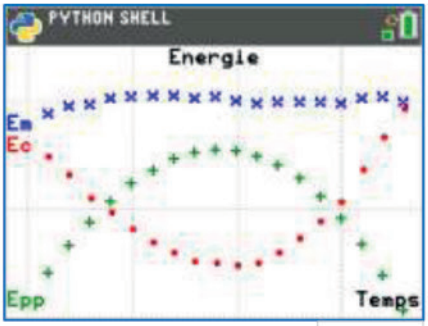



Fig. 5

## Fiche méthode

## Référentiel, compétences

**Capacités exigibles :**

- Exploiter les lois de Snell-Descartes pour la réfraction.
- Tester les lois de Snell-Descartes à partir d'une série de mesures.
- Déterminer l'indice de réfraction d'un milieu.

## Commentaires de l'auteur

Cette séance constitue une bonne opportunité de se familiariser avec la calculatrice TI-83 Premium CE Edition Python. La fiche sera ainsi plus détaillée, afin de prendre en main l'editeur python intégré à la calculatrice et d'avoir un aperçu des possibilités dans le traitement numérique.

Selon son énoncé (1), (2) ou (3), l'activité mettra l'accent sur l'une ou plusieurs *compétences des sciences expérimentales :*

- *Analyser/ Raisonner :*
  - ✓ Faire des prévisions à l'aide d'un modèle. (1)
  - ✓ Choisir un modèle ou des lois pertinentes. (2)
  - ✓ Choisir, élaborer, justifier un protocole. (3)
- *Réaliser :* Utiliser un modèle. (1), (2) et (3)
- *Valider :* Confronter un modèle à des résultats expérimentaux. (1) et (2)

*Compétences numériques :* pour chacun des énoncés suivants, il conviendra de programmer la fonction en langage python qui permettra de calculer la valeur de l'angle de réfraction $r$ en fonction de celui d'incidence $i$.

## Materiel

- Calculatrice TI-83 Premium CE Edition Python
- Bloc hémicylindrique transparent d'indice de réfraction 1,47
- Rapporteur (mesure d'angles)
- Source laser.

# Thème: vision et image : Loi de Descartes

Niveau: Seconde

TI-83 Premium CE

python

*Eric Tixidor*

## Fiche méthode

## Enoncés

**Enoncé 1 :** On donne la loi de la réfraction de Descartes :

$$n_1 \times \sin i = n_2 \times \sin r$$

Vérifier sa validité à partir de vos mesures expérimentales. L'indice de réfraction $n2$ du bloc transparent est donné ($n_2 = 1.47$ pour les essais réalisés dans cette fiche). L'indice de réfraction de l'air est $n_1 = 1$.

**Enoncé 2 :** On donne plusieurs formulations de la loi de la réfraction. Utiliser le matériel à disposition pour réaliser la mesure de l'angle de réfraction, en fonction de l'angle d'incidence. Faire plusieurs essais. Choisir alors la bonne formulation parmi les différentes lois proposées. L'indice de réfraction $n_2$ du bloc transparent est donné ($n_2 = 1{,}47$). Celui de l'air : $n_1 = 1$.

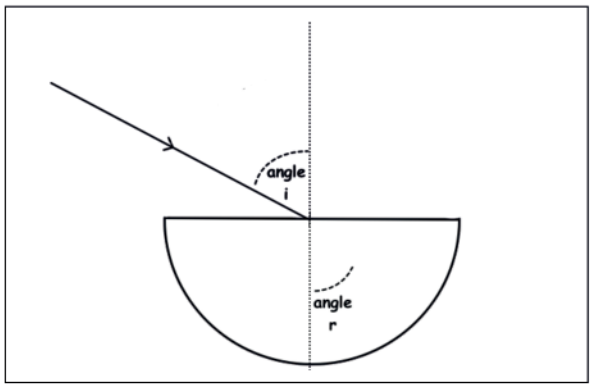
Loi 1 : $n_1 \times sin\, i = n_2 \times sin\, r$

Loi 2 : $n_1 \times i = n_2 \times r$

**Enoncé 3 :** A partir de la loi de Descartes, et à l'aide du matériel à disposition, déterminer la valeur de l'indice de réfraction $n_2$ du bloc transparent.

## Partie expérimentale

On utilise un bloc hémicylindrique constitué d'un matériau transparent, d'indice de réfraction $n_2 = 1.47$, qui repose sur un support permettant de mesurer les angles d'incidence et de réfraction de la lumière. Le rayon incident, issu d'une petite lampe au faisceau bien mince (laser), arrive sur le bloc par sa face rectiligne :




On réalise les mesures de l'angle de réfraction pour des angles d'incidence compris entre 0 et 90°.

Le rayon réfracté n'est pas représenté sur cette figure, mais se situe dans le quart d'espace opposé à celui du rayon incident.

## Résolution : Enoncé 1

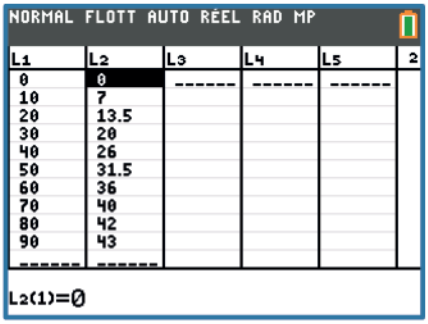
**1) Saisie des valeurs expérimentales**

Effacer les listes $L_1$, $L_2$ et $L_3$. En mode calculatrice,

Utiliser la combinaison de touches suivante :

2nde mém + `4:EffTtesListes`

Puis remplir les listes $L_1$ et $L_2$ avec les mesures réalisées : les angles $i$ seront saisis dans la liste $L_1$ ; les angles de réfraction $r$ dans la liste $L_2$ :




Faire listes stats 1 : modifier, puis remplir les colonnes $L_1$ et $L_2$ avec les valeurs.

**2) Calcul des valeurs attendue pour r à l'aide de l'application python**

**2.1) Principe :** On cherche à vérifier si les angles de réfraction suivent la loi de Descartes.

C'est à dire, si les valeurs de $r$ sont telles que :

$$\sin r \approx \frac{n_1 \times \sin i}{n_2}$$

Ou bien :

$$r \approx asin\left(\frac{n_1 \times \sin i}{n_2}\right)$$

Avec la fonction **sin** du module **math** du langage **python**, l'angle $i$ doit être exprimé en radians. Il faudra alors convertir la valeur lue par l'élève sur le rapporteur (degrés) à l'aide la fonction **radians** du module **math**.

La valeur retournée par **asin** est également en radians.

Pour un côté pratique, il sera nécessaire de convertir cette valeur en degrés avec la fonction **degrees** du module **math**.

Le terme $asin\left(\frac{n_1 \times sin\, i}{n_2}\right)$ se traduit en langage python par :

```
degrees(asin(n1*sin(radians(i))/
n2))
```

L'élève écrira une fonction **descartes** qui aura pour rôle de calculer la valeur de l'angle $r$ selon la loi de Descartes, à partir des paramètres $i$ (angle d'incidence en degrés), $n_2$ (indice de réfraction du milieu transparent) et $n_1$ (indice de réfraction de l'air, mis à **1** par défaut).

### 2.2) Saisie pas à pas du programme à la calculatrice :

Nous allons écrire le script **DESCART** ci-contre :


Sur la calculatrice, lancer l'application python :

dessin C
prgm **2:Python App**

Créer un nouveau programme que vous appellerez **DESCART** :
Nouv Nom = **DESCART**

Importer la librairie (le module) **math** :

Fns... › **Modul** **1:Math** puis **1:from math import ***

```
from math import *
```

Définir la fonction **descartes** et déclarer ses paramètres :

Fns... **1:def fonction( )**

```
def descartes(i,n2,n1=1):
```

Puis l'instruction de sortie de la fonction avec retour d'une valeur :

Fns... **2:return**

```
return
```

Compléter avec l'opération arithmétique vue plus haut.

Les fonctions **degrees** et **radians** se trouvent avec :

Fns... › **Modul** **1:Math** **Trig**

```
degrees(asin(n1*sin(radians(i))/
n2))
```

*Astuce pour la saisie : Alterner les claviers caractères et numériques avec les touches :* 2nde verr A alpha

**2.3) Utiliser la fonction `descartes` depuis le shell :**

On peut maintenant calculer les valeurs de l'angle $r$ en fonction de celles de $i$. Cela revient à :

- Executer le programme, charger les variables et fonctions en mémoire, et quitter l'éditeur et revenir dans le shell :
  commande [Exéc]

- Dans le shell, appeler la fonction **`descartes`** suivante. Cela aura pour effet de calculer $r$ pour un angle d'incidence $i = 30$ et une valeur $n$ pour l'indice de réfraction du bloc transparent :

```
descartes(30,n)
```

*Aide à la saisie :* La fonction **`descartes`**, et la variable **`n`** sont accessibles depuis le menu de la touche [distrib var].

- Le shell affiche alors la sortie suivante :
  La valeur de $n$ a été renseignée dans le script.
  La valeur renvoyée est proche de celle mesurée, compte tenu de la précision du rapporteur utilisé : $19.885.. \approx 20$

```
>>> # L'exécution de DESCART
>>> from DESCART import *
>>> descartes(30,n)
19.88516345460782
>>>
```

**3) Calcul des valeurs attendue pour r pour toutes les valeurs des angles i**

Pour vérifier la validité de la loi, il sera plus rigoureux de réaliser ce calcul pour *CHACUNE* des valeurs de $i$, et donc pour chacune des valeurs de la liste $L_1$. On aura besoin de faire communiquer l'application python avec les listes stockées dans la mémoire de la calculatrice.

Pour cela, on utilisera deux fonctions de la librairie **`ti_system`** :

- La fonction **`recall_list`** qui prend pour paramètre le **numero de liste** stockée dans la mémoire du mode calculatrice. Cette fonction retourne un objet de type liste, qui pourra être affectée à une nouvelle liste de l'application python.

- La fonction **`store_list`** qui copie les valeurs d'une liste python vers l'une des listes de la mémoire du mode calculatrice.

La fonction **store_list** sera appelée dans le shell de la manière suivante :

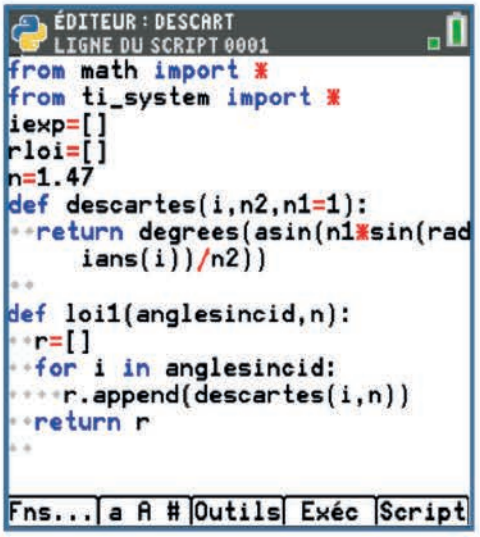
```
store_list("numero_de_liste",lis
te_python_à_copier)
```

Mais pour l'instant, poursuivons la saisie du script.
Il faudra créer une nouvelle fonction **loi1**, munie des paramètres **anglesincid** et **n** :

```
def loi1(anglesincid,n)
```

Cette fonction parcourt la liste mise en paramètre et calcule pour chacune d'entre elles la valeur donnée par la loi de réfraction. Cette valeur est ajoutée (**append**) à une liste locale, **r**, qui est renvoyée lorsque le parcours de la liste **anglesincid** est terminé.

**Script complet :**




*Aide pour la saisie :*
La méthode append associée aux listes se trouve avec :

Fns... > List

Il suffira alors d'affecter l'objet liste, retourné par la fonction **loi1,** à la liste **rloi** par exemple. (dans le shell)

- Exécuter le script, revenir dans le shell : Exéc, et saisir les deux lignes suivantes :

```
>>> iexp = recall_list("1")
>>> rloi = loi1(iexp,n)
```

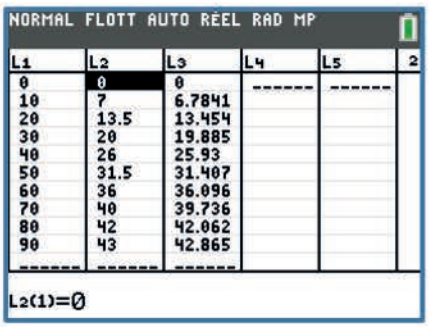
- Il restera alors à copier les valeurs de **rloi** dans l'une des listes du mode calculatrice, comme par exemple $L_3$ :

```
>>> store_list("3",rloi)
```

### 4) Comparer les valeurs expérimentales et celles calculées à partir de la loi

Quitter l'application python et retourner en mode calculatrice.
Pour vérifier que la liste $L_3$ a bien été complétée, afficher le tableau de valeurs avec :

listes
stats **1:modifier**




On peut observer cette fois, que ce sont TOUTES les valeurs des deux listes $L_2$ et $L_3$ qui sont très proches :

### 5) Affichage graphique

Pour confirmer que la loi définie dans la fonction **loi1** est une bonne modélisation des résultats expérimentaux, on peut tracer sur le même graphique les courbes :

- ✓ $L_2$ en fonction de $L_1$
- ✓ $L_3$ en fonction de $L_1$

*Paramétrage de la **première courbe** :*
Dans le menu graph stat, accessible depuis :

graph stats f1
2nde f(x) **1:Graph1…Aff** choisir :
Afficher la courbe : **Aff**

**XListe** : remplacer la liste par **$L_1$** à l'aide de la combinaison de touches :

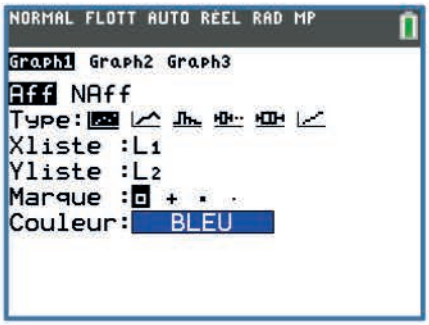
listes
2nde stats **1:$L_1$**

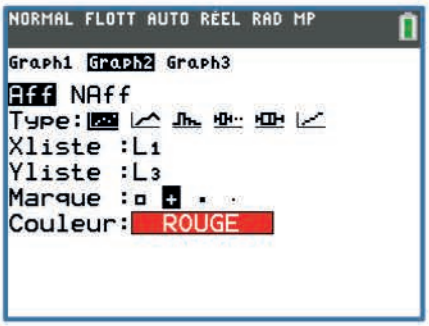
**YListe** : choisir **$L_2$**

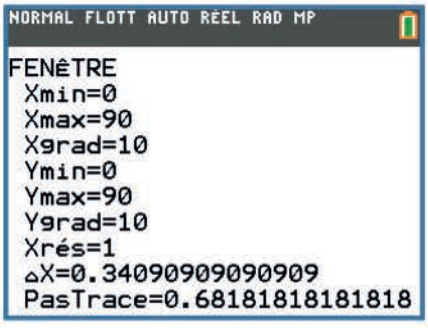
*Paramétrage de la **deuxième courbe** :*
**YListe : $L_3$**

*Modifier les axes :*

déf table f2
fenêtre puis modifier les valeurs selon l'image ci-contre :

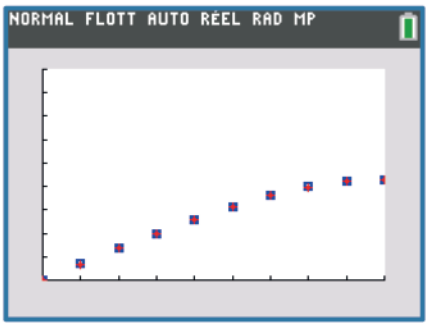



*Puis afficher le graphique :* table f5 graphe

On voit que les deux courbes (nuages de points) bleues et rouges se superposent bien. Le modèle suit bien les valeurs expérimentales relevées.



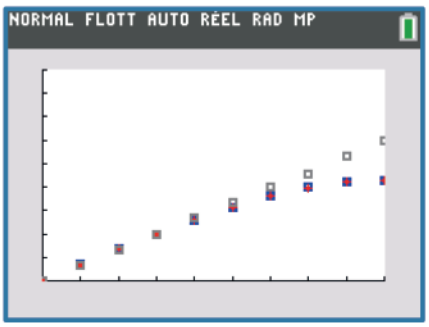

## Proposition de resolution : Enoncé 2

On peut ajouter une autre fonction **loi2** par exemple, qui calculera les valeurs issues d'un autre modèle, comme : $r = \frac{i}{1.5}$

```
def loi2(anglesincid,n=1.5)
  r=[]
  for i in anglesincid:
    r.append(i/n)
  return r
```

Les valeurs retournées par la fonction pourraient être copiées dans une quatrième liste, $L_4$. La représentation graphique de $L_4$ en fonction de $L_1$ montrerait alors que les points de $L_4$ (carré gris sur l'image), s'écartent de la représentation graphique de $L_2$ (valeurs expérimentales).

Cette loi ne correspond pas aux valeurs mesurées.



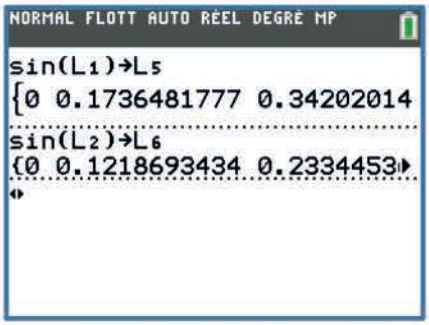

## Proposition de resolution : Enoncé 3

A partir des valeurs des angles i et r mesurés et stockés dans les listes $L_1$ (angles $i$) et $L_2$ (angles $r$) :
On utilisera les fonctionnalités de la calculatrice pour déterminer l'équation de la droite : $y = a \times x + b$
où **y = sin(i)** et **x = sin(r).**
Si les valeurs suivent la loi de Descartes, la valeur de **b** sera pratiquement nulle, et celle de **a** aura pour valeur : $n_2/n_1$ (soit le rapport des indices de réfraction du milieu transparent sur celui de l'air).
On aura pratiquement $a = n_2$.

**1) Calcul des sinus des angles i et r mesurés**

- Avec la touche quitter mode, mettre la calculatrice en *degrés.*

- stocker dans les listes $L_5$ et $L_6$ les valeurs :
  - ✓ $\sin(L_1)$ dans $L_5$
  - ✓ $\sin(L_2)$ dans $L_6$



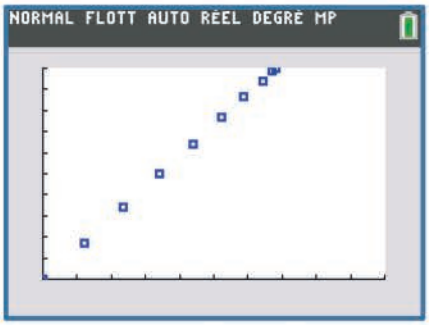

### 2) Vérification graphique

Tracer le graphique avec :

- ✓ **XListe : L6**
- ✓ **YListe : L5**

Modifier les échelles des axes avec :

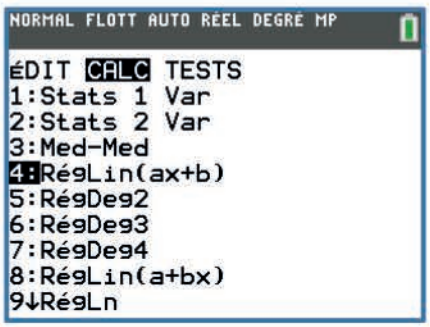
- ✓ **Xmax = 1**
- ✓ **Ymax = 1**




Le graphique doit montrer que les points (X,Y) sont alignés, en accord avec la loi de Descartes.

### 3) Modélisation

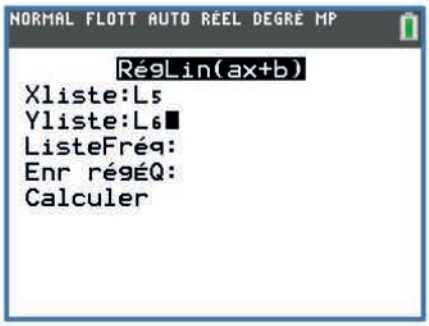
- listes stats **CALC 4 : RégLin(ax+b)**



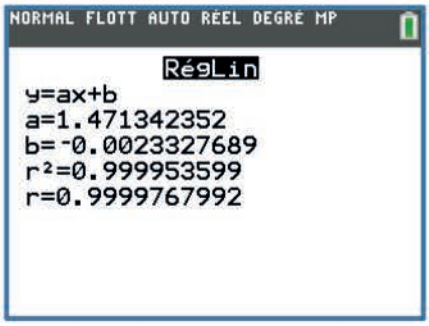

- Choisir :

✓ **XListe : L5**

✓ **YListe : L6**




- Puis descendre dans la fenêtre jusqu'à : **CALCULER**




Les coefficients a et b calculés sont bien en accord avec les valeurs attendues : b = 0 et **a = 1.47** (valeur de l'indice de refraction du bloc transparent).

Documents et script à télécharger à l'adresse :
https://education.ti.com/fr/physique-chimie

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Thème : Découverte des dispositifs intégrés du Ti-Innovator™ Hub : **DEL RGB, DEL rouge, Haut-parleur, Capteur de luminosité.**

**TI-83 Premium CE**

python™

**Stéphane Bonnaud**

# Fiche méthode

## Référentiel, compétences

- **Capacité** : prendre en main les dispositifs intégrés du microcontrôleur TI-Innovator™ Hub, à l'aide du langage de programmation Python.

## Commentaires de l'auteur


Fig. 1

- Le TI-Innovator™ Hub de Texas Instruments (fig. 1) est une carte à microcontrôleur. Il peut être un outil privilégié pour aborder sereinement avec les élèves les capacités expérimentales numériques de physique-chimie.
- Le Hub est muni de 3 entrées (IN1, IN2 et IN3, fig.2, à gauche) et de 3 sorties (OUT1, OUT2, OUT3, fig.2, à droite), toutes programmables avec Python. Sur chaque port (IN ou OUT), il est possible de connecter un capteur (IN) ou un actionneur (OUT).
- Le Hub est également muni de **4 dispositifs déjà intégrés**, que nous allons détailler dans cette fiche méthode : un capteur de luminosité et les trois actionneurs (DEL RGB, DEL rouge et haut-parleur).

  Le Hub communique avec la calculatrice TI-83 premium CE grâce à un câble USB. Le langage de programmation utilisé dans cette fiche est Python.

Fig. 2



## Matériel

- Calculatrice TI-83 premium CE Edition Python.
- TI-Innovator™ HUB et le câble USB de liaison.

## Prérequis

- S'assurer que le Hub soit bien à jour. Pour cela suivre la procédure suivante :
  1. Télécharger le logiciel de mise à jour du TI-Innovator™ Hub
  2. L'installer sur votre ordinateur. Lancer le logiciel.
  3. À ce stade, si le Hub n'est pas relié à l'ordinateur le message ci-contre s'affiche (fig.3).
  4. Relier le HUB à l'ordinateur avec le câble USB approprié en utilisant l'entrée **POWER (PWR) du Hub.** Un message de confirmation apparaît (fig.4). Attention, il est recommandé d'utiliser le câble livré avec le Hub.
  5. Un menu propose alors de télécharger puis d'installer le fichier *sketch* le plus récent pour mettre à jour le Hub (1.4.0.28.Hub en avril 2020).


Fig. 3

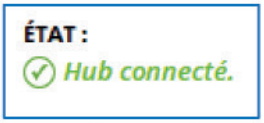
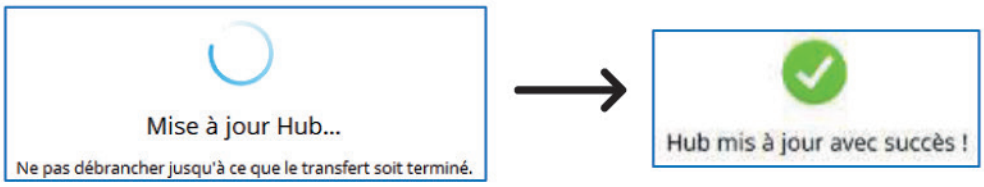



Fig. 4



## Avertissement

Nous allons présenter en détail comment utiliser les 4 dispositifs intégrés au Hub.

**Étape 1** : Prise en main *détaillée* de la diode RGB (actionneur).

**Étape 2** : Prise en main de la diode rouge (actionneur), moins détaillée qu'à l'étape 1, car il existe des similitudes avec l'étape 1.

**Étape 3** : Prise en main du haut-parleur (actionneur).

**Étape 4** : Prise en main du capteur de luminosité (capteur).

Le SCRIPT **HUB1** rassemblera les **4** fonctions nécessaires à la découverte de ces **4** dispositifs intégrés.

## Étape 1 : Découverte de la diode RGB : le module Color

- Une diode RGB programmable est intégrée au Hub (fig. 5). Elle permet d'obtenir une couleur au choix parmi $255^3$ =16,7 millions de couleur. Chaque couleur est caractérisée par un triplet de valeurs $(R, V, B)$.

  On souhaite faire briller la diode RGB selon la couleur de notre choix, par exemple en magenta, caractérisé par le triplet $(R, V, B) = (255,0,255)$. Cette diode est repérable sur le Hub par la mention *COLOR* (fig. 5).

Fig. 5 : La diode RGB a été ici recouverte d'un papier calque pour mieux diffuser la lumière.

- Préparer un nouveau script Python nommé HUB1.
- Dans le SCRIPT HUB1 :

1. Dans Modul, sélectionner le menu 6 : ti_Hub... (fig. 6)
2. Sélectionner 1 : Dispositifs intégrés du Hub... (fig. 7)
3. Sélectionner 1 : Color (fig. 8). Le module **`Color`** est ainsi importé au début du script (**`import color`**), voir fig. 11 ligne 2.
4. Dès lors, un nouveau module nommé **`Color`** apparaît dans la liste des modules disponibles (fig. 9). Toutes les fonctions du module **`Color`** sont alors accessibles dans le menu Modul avec 8 : Color...

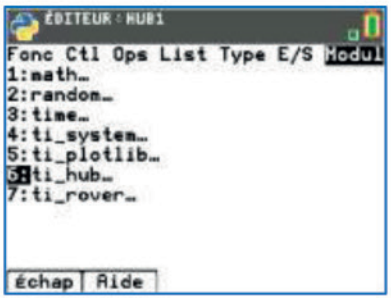
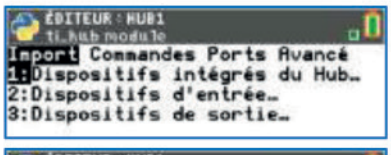



Fig. 6




< Fig. 7


< Fig. 8

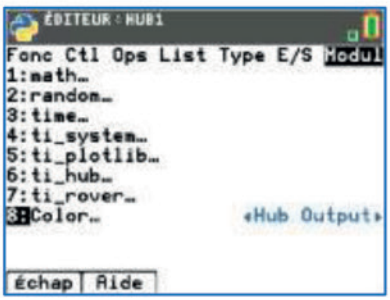



Fig. 9

5. Le module **Color** possède 3 fonctions : **rgb**, **blink** et **off**, (fig. 10).


Fig. 10

- La fonction **rgb** allume la DEL RGB avec la couleur correspondant au triplet **(r,g,b)**. Chaque variable **r**, **g** ou **b** a une valeur entière comprise entre 0 et 255.
- La fonction **blink** fait clignoter la DEL avec la fréquence **freq**, exprimée en Hz, dans l'intervalle [0.1 Hz ; 20 Hz] pendant une durée égale à **time**, exprimée en secondes.
- La fonction **off** éteint la DEL RGB. Attention, cette fonction éteint la DEL immédiatement, sans attendre par exemple qu'elle ait fini de clignoter. Si besoin, un appel à la méthode **sleep(secondes)** du module **time** permettrait de temporiser en mettant le programme sur pause quelques instants.

- Écrire une fonction Python **couleur**, munie des 5 arguments **r,g,b,f,dt**, qui fasse clignoter la DEL avec la couleur **(r,g,b)** souhaitée, à la fréquence **f,** pendant la durée **dt** (fig. 11).


Fig. 11

## Étape 2 : Découverte de la diode Rouge : le module Light

- Sur le même modèle que le module **Color**, le module **Light** permet de piloter la DEL rouge du Hub (fig. 12). Cette DEL est toujours de couleur rouge.


Fig. 12 : La diode rouge a été ici recouverte d'un papier calque pour mieux diffuser la lumière.

- Dans le SCRIPT HUB1 :
  1. Importer le module **Light** : **import light** s'affiche au début du script (fig. 11, ligne 4).
  2. Le module **Light** possède 3 fonctions : **on, off** et **blink**, dont les rôles sont explicites (fig. 13).

  NB : **freq** appartient à l'intervalle [0.1 Hz ; 20 Hz].

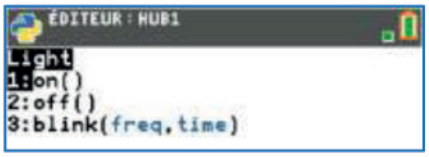



Fig. 13

- Écrire une fonction Python **rouge**, munie de 2 arguments **f** et **dt**, qui fasse clignoter la DEL rouge à la fréquence **f,** pendant la durée **dt** (fig. 14).

Fig. 14

```
def rouge(f,dt):
    light.blink(f,dt)
```

# Étape 3 : Découverte du haut-parleur : le module Sound

- Sur le même modèle que le module **Color**, le module **Sound** permet de produire un son à l'aide du haut-parleur intégré au Hub (fig. 15).


Fig. 15

- Dans le SCRIPT HUB1 :
  1. Importer le module **Sound** : `import sound` s'affiche au début du script (fig. 11, ligne 3).
  2. Le module **Sound** possède 2 fonctions : **tone** et **note** dont les rôles sont explicites (fig. 16).

NB : L'argument `time,` de type *float*, est optionnel, compris entre 0.1 et 100.0, exprimé en secondes.

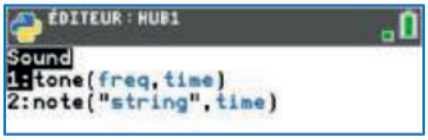



Fig. 16

- ❖ La fonction **tone** produit un son pur de fréquence `freq`, exprimée en Hz, dans la gamme [1Hz ; 8000 Hz] pendant la durée `time`, exprimée en secondes.
- ❖ La fonction **note** produit une note de musique à partir de son nom, codé selon la norme anglo-saxonne, qui utilise des notes de l'alphabet. Chaque note est suivie d'un chiffre correspondant à son octave. Ainsi la note « La 440 », qui est un $La_3$, s'écrit en notation anglo-saxonne "`A4`".

| Notation Française | Notation Anglo-saxonne |
|---|---|
| Do | C |
| Ré | D |
| Mi | E |
| Fa | F |
| Sol | G |
| La | A |
| Si | B |

- Écrire une fonction Python **joue1,** munie des arguments **f et dt,** qui joue une note de fréquence **f**, pendant la durée **dt** (fig. 17).

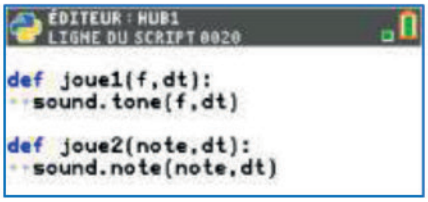



Fig. 17

- Écrire une fonction Python **joue2**, qui joue une note appelée `note` en notation anglo-saxonne (ne pas oublier les guillemets lors de l'appel de la fonction) pendant la durée `dt` (fig. 17). L'argument `note` est de type chaîne de caractère.

# Étape 4 : Découverte du capteur de luminosité : le module Brightness


Fig. 18

- Sur le même modèle que le module **Color**, le module **Brightness** permet de mesurer une luminosité à l'aide du capteur de luminosité intégré (fig. 18).

- Dans le SCRIPT HUB1 :
  1. Importer le module **Brightness** : `import brightns` s'affiche au début du script (fig. 11, ligne 5).
  2. Le module **Brightness** possède 2 fonctions : **measurement** et **range**, fig. 19.


Fig. 19

  - ❖ La méthode **measurement** déclenche la mesure de la luminosité. Son unité n'est pas connue. La valeur renvoyée est un nombre flottant compris entre 0% (très sombre) et 100% (très lumineux).
  - ❖ La méthode **range** permet de définir si nécessaire une plage de mesure, comprise entre `min` et `max`.

- Écrire une fonction Python **lumi** qui renvoie la valeur de la luminosité mesurée par le capteur, comprise entre 0 et 1000. (fig. 20).

```
def lumi():
  brightns.range(0,1000)
  mes=brightns.measurement()
  return mes
```

Fns… | a A # | Outils | Exéc | Script

Fig. 20

Télécharger le script **HUB1** à l'adresse : https://education.ti.com/fr/physique-chimie.

## Référentiel, compétences

- **Capacité** : prendre en main les dispositifs intégrés du microcontrôleur TI-Innovator™ Hub (fig. 1), à l'aide du langage de programmation Python.
- Utiliser un dispositif avec microcontrôleur et capteur.
- Passer de l'algorithme au programme.


Fig. 1

## Commentaires de l'auteur

- Une carte à microcontrôleur dispose d'entrées et de sorties programmables à l'aide d'un langage de programmation. Le Hub est une carte programmable en Python qui dispose de 4 dispositifs intégrés : un capteur de luminosité et trois actionneurs (DEL RGB, DEL rouge et haut-parleur).
- Nous allons découvrir dans cette fiche-méthode deux sketches destinés à faciliter la prise en main de la calculatrice TI-83 Premium CE Edition Python et du microcontrôleur TI-Innovator™ Hub.
- Le premier sketch propose de jouer un son d'autant plus aigu que la luminosité ambiante est élevée avec :
  - ✓ En entrée : le capteur de luminosité intégré au Hub.
  - ✓ En sortie : un actionneur, le haut-parleur intégré au Hub.
- Le deuxième sketch propose de simuler le contrôle de la luminosité d'un écran de smartphone. L'écran sera d'autant plus lumineux que la luminosité ambiante est élevée :
  - ✓ En entrée : le capteur de luminosité intégré au Hub.
  - ✓ En sortie : un actionneur, la diode RGB intégrée au Hub.

## Matériel

- TI-83 Premium CE Edition Python.
- TI-Innovator™ Hub et le câble USB de liaison.

## Prérequis

- Savoir utiliser les dispositifs d'entrée et de sortie intégrés au Hub. Consulter si besoin la fiche méthode intitulée « Découverte des dispositifs intégrés du TI-Innovator™ Hub ».

# PREMIER SKETCH : Jouer un son d'autant plus aigu que la luminosité ambiante est élevée

- **OBJECTIF** : Grâce à la programmation, nous allons déclencher une action (modifier la fréquence d'un son) en fonction de la mesure d'une luminosité. La note jouée sera dans une gamme comprise entre un $La_3$ et un $La_4$ :

| Note jouée | % de luminosité mesuré |
|---|---|
| $La_3$ $(220\ Hz)$ | 0 % de luminosité |
| $La_4$ $(440\ Hz)$ | 100 % de luminosité |

- **ALGORITHME**

Tant que vrai :

$lum \leftarrow$ Mesure de la luminosité ambiante

$f \leftarrow$ calcul de la fréquence du son à émettre en fonction de la valeur de $lum$

Émettre le son de fréquence $f$

Fin de tant que.

- **FONCTIONS PYTHON**

1) Créer un SCRIPT nommé SKETCH1.
2) L'éditer et écrire une fonction Python **`lumi`**, qui renvoie la valeur de la luminosité ambiante, mesurée avec le capteur de luminosité intégré au Hub.
3) Écrire une fonction Python **`note`**, qui joue le son de fréquence $f$ pendant $0.2\ secondes$.
4) On souhaite que la fréquence du son joué soit une fonction affine de la luminosité. Trouver l'expression de la fréquence $f$ en fonction de la luminosité $lum$ mesurée.
5) À l'aide des fonctions précédentes, écrire une fonction Python **`joue`**, qui émet un son dont la fréquence est fonction de la luminosité ambiante mesurée.

- **SOLUTION PROPOSÉE**

$f = 2{,}2 \times lumi() + 220$

```
ÉDITEUR : SKETCH1
LIGNE DU SCRIPT 0001
import sound
import brightns
def lumi():
  lum=brightns.measurement()
  return lum
def note(f):
  sound.tone(f,0.2)
def joue():
  while True:
    f=2.2*lumi()+220
    note(f)
Fns… | a A # | Outils | Exéc | Script
```

Télécharger le programme **SKETCH1** à l'adresse :

https://education.ti.com/fr/physique-chimie

# DEUXIÈME SKETCH : Rendre l'écran d'un smartphone d'autant plus lumineux que la luminosité ambiante est élevée

- **OBJECTIF** : Grâce à la programmation, nous allons déclencher une action (modifier l'éclat d'une DEL qui modélisera l'écran lumineux) en fonction de la mesure d'une luminosité.
  - ✓ On choisit ici de travailler avec la DEL RGB, dont la couleur est déterminée par le triplet **(r,g,b)** avec **r**, **g** et **b** trois entiers compris entre 0 et 255. En faisant varier les valeurs du triplet entre **(0,0,0)** et **(255,255,255)**, il est possible de moduler l'éclat de la DEL.

- **ALGORITHME**

Tant que vrai :

$lum \leftarrow$ Mesure de la luminosité ambiante

$val \leftarrow$ calcul d'une valeur en fonction de la valeur de $lum$

Affecter la valeur $val$ à la diode RGB pour qu'elle s'éclaire plus ou moins.

Fin tant que.

- **FONCTIONS PYTHON**

1) Créer un SCRIPT nommé SKETCH2.
2) L'éditer et écrire une fonction Python **couleur**, munie de l'argument **val**, qui éclaire la DEL RGB avec le triplet **(val,val,val)**.
3) Écrire une fonction Python **lumi**, qui renvoie la valeur de la luminosité ambiante, mesurée avec le capteur de luminosité intégré au Hub.
4) On souhaite que la valeur **val** soit une fonction linéaire de la luminosité renvoyée par la fonction **lumi**. Trouver l'expression de **val** en fonction de la luminosité mesurée.
5) À l'aide des fonctions précédentes, écrire une fonction Python **ecran**, qui éclaire la DEL en fonction de la luminosité ambiante mesurée.

- **SOLUTION PROPOSÉE**

$val = 2{,}55 \times lumi()$ . Ainsi si **lumi** renvoie 0%, **val** vaudra **0** et si **lumi** renvoie 100%, **val** vaudra **255**.

Remarque : en réalité, un écran de smartphone n'est jamais complètement éteint ! On peut alors définir une luminosité minimale « seuil » égale à 5 grâce à une fonction **max**. Pour cela remplacer la dernière ligne du script par :

```
    couleur(max(val,5))
```


Télécharger le programme **SKETCH2 à l'adresse** :

https://education.ti.com/fr/physique-chimie.

# Fiche méthode

*Eric Tixidor*

## Référentiel, compétences

**Capacités exigibles :**

- Utiliser un dispositif comportant un microcontrôleur pour produire un signal sonore.
- Réaliser une série de mesures.

## Commentaires de l'auteur

On ajoutera pour cette séance la création d'un télémètre à ultra-sons qui s'ajoutera au dispositif sonore.

- Capacités numériques : Utiliser les entrées et sorties d'un microcontrôleur, en langage python.
- Capacités mathématiques : définir et programmer une fonction affine et une fonction racine carrée.

## Matériel

- Calculatrice TI-83 Premium CE Edition Python
- Le microcontrôleur TI-Innovator™ Hub et son haut-parleur intégré.
- Un émetteur-récepteur à ultra-sons avec connectique groove, qui sera branché sur le port **`IN 1`** du Hub.

## Enoncé

On souhaite créer un dispositif autonome de type « radar de recul ». Lorsque la distance à un obstacle diminue, ce dispositif devra non seulement émettre un son de fréquence plus aiguë, mais aussi avec des bips plus rapprochés.

Enfin, il serait préférable que ces variations de fréquences sonores soient plus marquées pour des obstacles se situant à moins de 1,0m (c'est à dire, entre la distance de contact et 1,0m). Les variations sonores pourront être moins importantes pour de plus grandes distances.

## Prérequis

Le lecteur de cette fiche devra être déjà familiarisé avec l'environnement python de la calculatrice.

# Proposition de résolution

## Mesurer une distance avec un télémètre à ultra-sons

On utilisera le capteur de type émetteur-récepteur à ultra-sons, que l'on branchera sur le port **IN1** du TI Innovator Hub.

Dans l'éditeur python de la calculatrice : créer un nouveau script que l'on appellera : **SON**
Importer les librairies nécessaires pour :

- Utiliser le capteur à ultra-son :
  Modules **6:ti_hub** puis **2:dispositifs d'entrée**
  **2:Ranger** ce qui renvoie : **from ranger import ***
- Ecrire une boucle non bornée sur la fonction escape( ) : Modules
  **4:ti_system** puis **1:from ti_system import ***
- Ecrire dans une fenêtre graphique sur l'écran de la calculatrice :
  Modules **5:ti_plotlib** puis 1:**import ti_plotlib as plt**

```
from ti_system import *
import ti_plotlib as plt
from ranger import *
```

On définit une fonction **screen** qui permettra un affichage dans la fenêtre graphique à l'aide de la librairie **ti_plotlib**.
Cette fonction prendra en paramètre :

- ✓ **y** : Le numéro de la ligne à laquelle le message s'affichera
- ✓ **val** : la valeur numérique qui complète le texte affiché
- ✓ **msg** : le texte à afficher, qui comprend une règle de formatage lui permettant de remplacer les caractères **%.3f** (réel) ou **%.0f** (entier) par **val**.

```
def screen(y,val,msg):
    plt.text_at(y,msg%val,
    "center")
```

Saisir ensuite les instructions qui permettront :

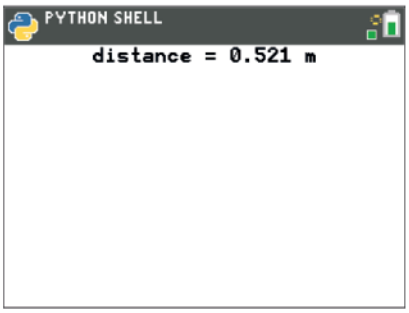
- ✓ de déclarer l'objet associé au port groove **IN 1**.
- ✓ d'effacer l'écran

```
sr=ranger("IN 1")
plt.cls()
```

On complète enfin le corps du programme avec une boucle non bornée qui s'exécute tant que l'utilisateur n'appuie pas sur la touche off on.
Dans cette boucle, le programme permet de relever les mesures de distance, en continu, et avec un affichage sur l'écran de la calculatrice.

```
while not escape( ):
    d = sr.measurement( )
    msg="distance = %3.f m"
    screen(1,d,msg)
    sleep(0.2)
```

Brancher l'émetteur-capteur à ultra-sons sur le port **IN 1** du Hub.
Diriger vers un obstacle et mesurer la distance.
La calculatrice affiche sa distance en mètre.




La plage de mesure annoncée par la notice du capteur est de 2cm – 400cm avec une résolution de 1cm.

Les valeurs retournées semblent conformes.

*Remarque :* Il peut y avoir parfois quelques *exceptions*, et la valeur renvoyée peut être soudainement égale 10,0m. Il faudra considérer ce point pour le calcul de la fréquence sonore.

## Fonction de calcul de la fréquence sonore

Le haut-parleur intégré émet des sons de fréquences 200 à 4500 Hz environ. On cherche une fonction qui fait correspondre les valeurs de distances avec celles des fréquences, jusqu'à d = 1,0m.
Les obstacles seront jugés préoccupant lorsque la distance est inférieure à 1m.

Cette fonction doit être décroissante si l'on veut une ***fréquence plus élevée*** (son plus aigu) lorsque ***l'obstacle est rapproché.***

- Une première idée serait de programmer une fonction **freq** affine à partir des correspondances suivantes :

| d(cm) | 0 | 100 |
|---|---|---|
| f(Hz) | 4500 | 200 |

On pourrait alors définir la fonction suivante :

```
def freq(d):
    return 4500-4300*d
```

Et, afin d'éviter de retourner des valeurs négatives pour **f** (voir la remarque plus haut sur les *exceptions* qui génèrent des valeurs de d>1), on ajoutera la ligne suivante :

```
if d>1 : d = 1
```

- Une deuxième idée serait de définir une fonction **freq**, dont les *variations* seraient plus importantes pour des plus petites valeurs de d. Cela créé un effet *critique* lorsque l'obstacle se rapproche.

Ce serait alors une fonction non linéaire en **d** (c'est-à-dire dont la représentation graphique n'est pas une droite). Une possibilité est d'utiliser la racine carrée de d :

```
def freq(d):
    if d>1 : d=1
    return 4500-4300*d**0.5
```

On pourra saisir cette deuxième fonction à la suite des imports de librairies dans le programme.

## Programmer la sortie sonore

Dans l'éditeur du script, importer la librairie nécessaire pour :

- Utiliser haut-parleur intégré : **Modules** **6:ti_hub** **1:dispositifs intégrés du Hub** puis **3:Sound**

```
import sound
```

On ajoutera alors dans la boucle non bornée :

- L'instruction qui permet de calculer la fréquence sonore en fonction de la distance mesurée :

```
f = freq(d)
```

- La chaine formatée associée à la variable msg2 :

```
msg2 = "freq = %.0f Hz"
```

- L'appel de la fonction **screen** qui permettra d'afficher la fréquence sonore jouée par le Hub.

```
screen(2,f,msg2)
```

- L'instruction qui permet de jouer un son :

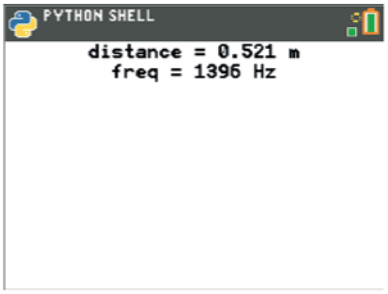
```
sound.tone(frequence,duree)
```

La fonction **sound.tone** prend 2 paramètres :

- ✓ la fréquence sonore.
- ✓ la durée du son. Avec une durée de 0.1s, le son ressemble plus à un Bip.

Enfin, on modifiera le paramètre de la fonction **sleep** en écrivant **sleep(d/2) :** c'est une valeur croissante avec la distance. Cela rajoutera à l'effet « *critique* » lorsque l'obstacle se rapproche, les Bips étant alors plus rapprochés.

Le programme mesure alors la distance, joue un son (ou plutôt une série de Bips sonores), et affiche la distance et fréquence de la note jouée.



La durée qui sépare deux Bips sonores (c'est à dire la période d'émission des Bips sonores) est plus courte lorsque l'obstacle est plus proche.

## Script complet

ÉDITEUR : SON
LIGNE DU SCRIPT 0001

```
from ti_system import *
import ti_plotlib as plt
from ranger import *
import sound

def screen(y,val,msg):
  plt.text_at(y,msg%val,"center"
      )
def freq(d):
  if d>1:d=1
  return 4500-4300*d**0.5

sr=ranger("IN 1")
plt.cls()

while not escape():
  d=sr.measurement()
  msg="distance = %.3f m"
  screen(1,d,msg)
  f=freq(d)
  msg2="freq = %.0f Hz"
  screen(2,f,msg2)
  sound.tone(f,0.1)
  sleep(d/2)
```

Fns… | a A # | Outils | Exéc | Script

Documents et script à télécharger à l'adresse :
https://education.ti.com/fr/physique-chimie

## Référentiel, compétences

- Capacité : Utiliser un dispositif avec microcontrôleur et capteur.
- Utiliser un langage de programmation.
- Prendre en main un capteur de pression Grove (fig.2) connecté sur l'entrée **IN1** du Hub (fig. 1).


Fig. 1

## Commentaires de l'auteur

- Le Hub est une carte à microcontrôleur disposant d'entrées et de sorties déjà intégrées programmables à l'aide d'un langage de programmation. Il est également possible d'utiliser une large liste de capteurs additionnels grâce aux entrées IN 1, IN 2 et IN 3.
- Nous allons découvrir dans cette fiche-méthode comment effectuer une mesure de pression avec un capteur de pression (fig. 2) relié au TI-Innovator™ Hub, lui-même connecté à une calculatrice TI-83 Premium CE Edition Python.
- Nous verrons ensuite un programme permettant de vérifier expérimentalement la loi de Mariotte $P \times V = constante$.

Fig. 2

## Matériel

- Calculatrice TI-83 premium CE Edition Python.
- TI-Innovator™ Hub et le câble USB de liaison.
- Kit de mesure de pression Grove MPX5700AP, avec tube silicone et seringue étanche. Il permet des mesures comprises entre 15 $kPa$ et 700 $kPa$.

## Prérequis

- Savoir utiliser les dispositifs d'entrée et de sortie intégrés au Hub. Consulter si besoin la fiche méthode intitulée « Découverte des dispositifs intégrés du TI-Innovator™ Hub ».

# Premières mesures de pression

**OBJECTIF** : Mesurer la pression atmosphérique avec le dispositif complet. Le capteur Grove doit être étalonné pour afficher une pression en Pascal ou en kPa.

- Rappels :

✓ $1\ bar = 10^5\ Pa = 10^3\ hPa = 10^2\ kPa$

✓ $1\ hPa = 1\ mbar$

✓ La pression atmosphérique normale est voisine de $1013\ hPa$.

- FONCTIONS PYTHON

1) Créer un SCRIPT nommé PRESSION dont le type est 6 : Projets STEM Hub (fig.3).

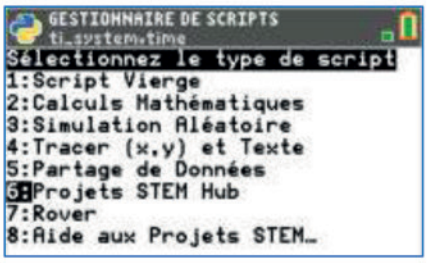



Fig. 3

2) Importer le module analogique du Hub à l'aide du menu 2 : Dispositifs d'entrée… (fig.4a et 4b)

Fig. 4a

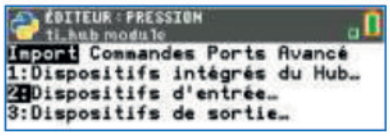
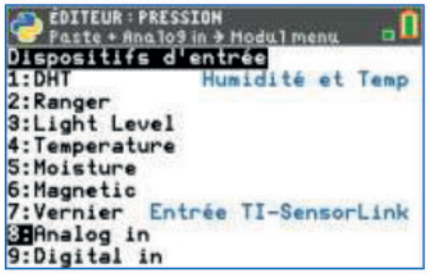



Fig. 4b




3) Écrire une fonction Python **p**, (fig.5), qui :

   a. Configure le port **IN 1** du Hub en tant que dispositif d'entrée.

   b. Effectue la mesure de la valeur de la pression atmosphérique.

   c. Renvoie la pression **p1** exprimée en hPa. Les deux valeurs **0.04628** et **-30.26** sont empiriques et peuvent être modifiées si nécessaire. Elles sont nécessaires pour que la pression mesurée soit exprimée en $hPa$.

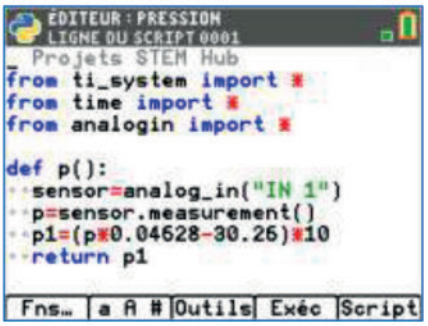



Fig. 5

4) L'appel à la fonction **p** renvoie la valeur de la pression atmosphérique, exprimée en hPa. (fig. 6).

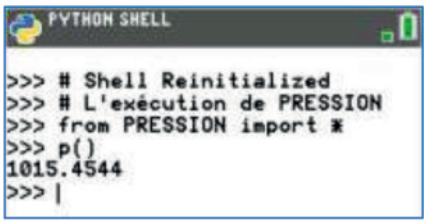



Fig. 6

# Vérification de la loi de Mariotte : $P \times V = constante$

**OBJECTIF**

La loi de Mariotte s'énonce ainsi : « Au cours de l'évolution isotherme d'un gaz parfait, le produit $P \times V$ est constant ». La loi a d'abord été découverte en 1662 par l'Irlandais Robert Boyle puis par le Français Edme Mariotte en 1676.

Dans cette expérience, l'air sera assimilé à un gaz parfait.

**PROTOCOLE EXPÉRIMENTAL**

En faisant varier le volume $V$ d'un gaz enfermé dans une seringue, on mesure sa pression à température constante.

- ✓ Le volume $V$ de gaz sera lu sur la graduation de la seringue, très imprécise.
- ✓ La pression $P$ du gaz sera mesurée avec le capteur de pression.
- ✓ On tracera ensuite la courbe $P = f(1/V)$.

Pour les besoins de l'expérience, la fonction **`input`** est *exceptionnellement* utilisée car elle permet une saisie au clavier du volume d'air enfermé dans la seringue, au cours de la détente du gaz. Il faut en effet un signal qui demande à $n$ reprises à l'expérimentateur de tirer sur le piston de la seringue, afin de dilater le gaz.

Voici les captures d'écran du programme à utiliser (fig. 7, 8 et 9) :


Fig. 7

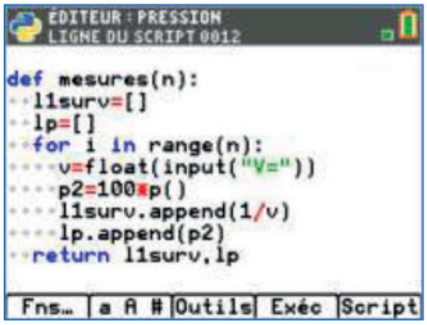



Fig. 8

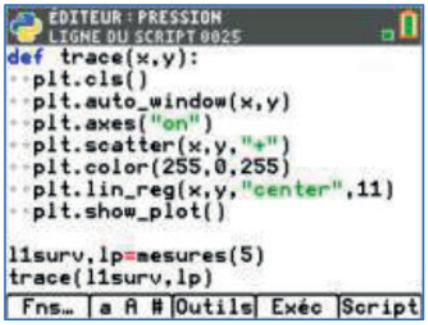



Fig. 9

**COMMENTAIRES**

Figure 7 : C'est le programme réalisé à la première partie (fig. 5).

Figure 8 : La fonction **`mesures`** va répéter $n$ fois la mesure de la pression. Deux listes de valeurs sont alors construites :

- La liste **`lp`** qui contient les $n$ valeurs successives de la pression, exprimée en $Pa$ (cf **`p2=100*p()`** )
- La liste **`l1surv`** qui contient les $n$ valeurs successives de $\frac{1}{V}$, exprimées en $mL^{-1}$.

Les deux instructions **`l1surv,lp=mesures(6)`** et **`trace (l1surv,lp)`** sont les deux appels de fonction qui rendent les mesures, puis leur tracé (avec régression linéaire) automatiques, dès le lancement du programme.

Voici les résultats obtenus pour une expérience de 6 mesures de volume (fig. 10 et fig. 11) :

- Le volume intérieur du tuyau est évalué à 3,4 mL (voir fig.12)
- Les volumes choisis sont les suivants :

| $V\ (mL)$ | $13{,}4\ =\ 10+3{,}4$ | 15,4 | 17,4 | 19,4 | 21,4 | 23,4 |
|---|---|---|---|---|---|---|

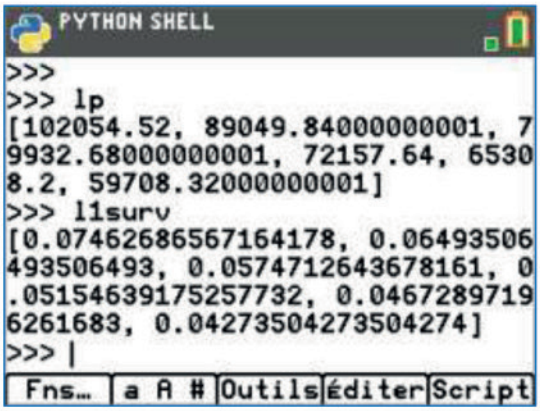



Fig. 10

Fig. 12 : sur cet exemple, le piston est sur la graduation 12 mL, ce qui correspond à un volume total d'air égal à 15,4 mL (12+3,4=15,4 mL)

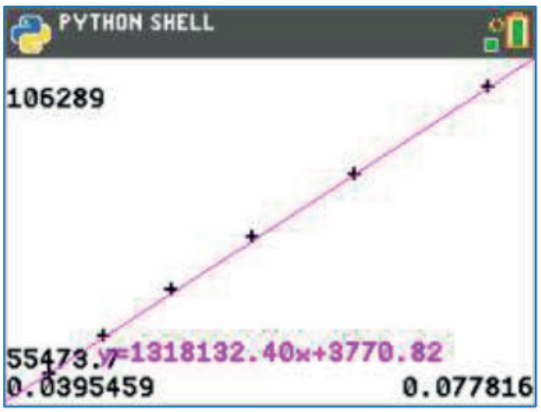



Fig. 11

**CONCLUSION**

La courbe $\boldsymbol{P = f\left(\frac{1}{V}\right)}$ est une droite passant par l'origine du repère (fig.11). Le produit $P \times V$ est donc constant, aux incertitudes de mesure près, qui n'ont pas été évaluées ici. On aurait pu également tracer $P \times V$ en fonction de V, pour obtenir une fonction constante.

Télécharger le script PRESSION à l'adresse : https://education.ti.com/fr/physique-chimie.

# Fiche méthode

## Référentiel, compétences

**Capacités exigibles :**

- Mesurer une tension et une intensité.
- Représenter et exploiter la caractéristique d'un dipôle.
- Capacités numériques : Représenter un nuage de points associés à la caractéristique d'un dipôle et modéliser la caractéristique de ce dipôle à l'aide d'un langage de programmation.

## Commentaires de l'auteur

Cette séance peut constituer une première approche de l'environnement constitué du TI-Innovator™ Hub et de la calculatrice TI-83 Premium CE Edition Python. Le programme qui consiste à piloter l'alimentation d'une branche avec une DEL amène à se familiariser progressivement avec l'éditeur python de la calculatrice. Et l'appropriation du breadboard sera essentielle pour réaliser des projets plus complexes.
Au menu dans cette séance :

- réaliser un circuit série simple, constitué d'une seule branche. On y fera les mesures de tension (à l'aide de l'une des entrées « analogiques » du TI-Innovator™ Hub) et de courant (ampèremètre).
- puis relever les valeurs de tension-courant qui permettront de tracer et modéliser la caractéristique d'un dipôle.

## Matériel

- Calculatrice TI-83 Premium CE Edition Python.
- Le microcontrôleur TI-Innovator™ Hub est utilisé avec les connecteurs de la platine (breadboard) :

La platine breadboard du Hub comprend :
- ✓ 2 ports d'alimentations (en bas à droite, voir image plus bas)
- ✓ Une série de ports reliés à la masse (0V) du Hub : sur la rangée inférieure.
- ✓ des entrées analogiques en 14 bits (BB5 BB6 ou BB7)
- ✓ et des sorties numériques (les autres BB).

Ce sera l'une de ces sorties numériques (BB5) qui fournira la tension d'alimentation (5V), que l'on pourra programmer à 0V ou 5V.
Les limitations du Ti-innovator™ Hub imposent une limitation du courant

- Diode électroluminescente DEL d'intensité du **courant nominal 10mA**, plaquette d'essais, fils.
- Conducteurs ohmiques, de résistances 1kΩ et 10kΩ.
- En option : un ampèremètre.

**Thème: Signaux et capteurs : mesure de tensions et de courants, caractéristique d'un dipôle**

Niveau: Seconde

TI-83 Premium CE

python

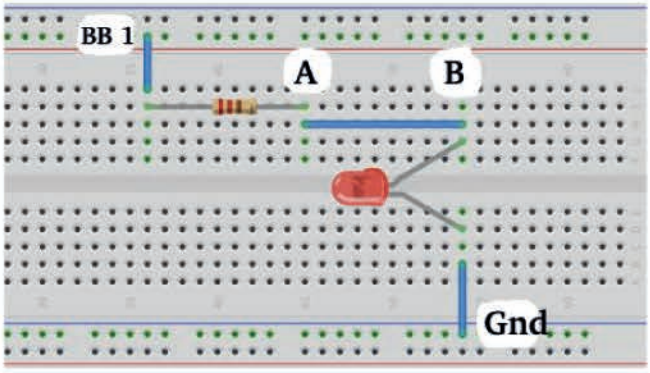
*Eric Tixidor*

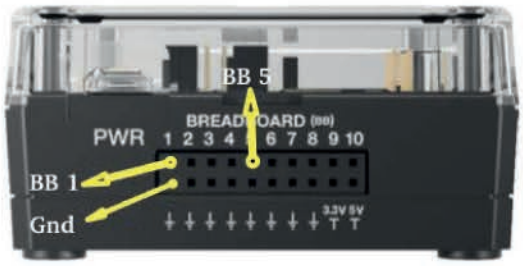
## Fiche méthode

## Enoncé

## Partie 1 : Allumer une diode électroluminescente

Allumer une DEL à l'aide de l'alimentation du Hub, et *choisir la résistance* de protection la mieux adaptée, c'est à dire celle qui donnera *le meilleur éclairement* compte tenu de la limitation du hub (5 mA max).
L'image suivante servira de support pour réaliser le circuit électronique.




L'**alimentation** sera fournie par la sortie **BB1** du Hub. La ligne connectée à **BB 1** sur la plaquette d'essai (à gauche) sera reliée au PIN 1 du breadboard du Hub (image de droite).

La ligne connectée à **Gnd** sera reliée à l'une des masses du breadboard (un des PIN de la **rangée inférieure**, à l'exception des 2 derniers à droite, réservés aux alimentations ; image de droite).

La résistance sera choisie parmi les deux à disposition : 1000Ω ou 10 000Ω.

## Partie 2 : Etude de la caractéristique d'un dipôle : La DEL

Pour étudier la **caractéristique intensité-tension** d'un dipôle, on met habituellement ce dipôle en série avec un rhéostat (un conducteur ohmique de résistance variable).

Pour cette expérience, on utilisera un jeu de différentes résistances dont les valeurs s'étalent de 1kΩ à 33kΩ à la place du rhéostat. Ce matériel est plus souvent présent dans nos laboratoires de physique.

Adapter alors le montage précédent, pour mesurer les valeurs de tension $U_{B\text{-}GND}$ (notée $U_B$) aux bornes de la DEL, ainsi que le courant I dans la branche.

**Permuter la résistance** de la branche par une autre, de valeur différente. Réaliser alors ces mêmes mesures de tension et de courant. Et **étudier la caractéristique de la DEL** à partir des couples de valeurs **($U_B$, I).**

# Proposition de résolution : partie 1

## 1) Réalisation du circuit

Utiliser une plaquette d'essais pour réaliser le circuit. Disposer la résistance et la DEL. Effectuer les branchements entre les composants et ajouter ceux vers les entrées/sorties BB du Hub.
Connecter le port DATA situé sur la face opposé du breadboard du Hub, à la calculatrice. Mettre celle-ci en marche.

## 2) Programmer la calculatrice

Le script suivant permet de fournir une tension fixe sur la sortie numérique BB1 :

Sur la calculatrice TI-83 Premium CE Edition Python, lancer l'application python :

dessin C
prgm **2:Python App**

Créer un nouveau script que vous appellerez HUB :

Nouv **Nom = HUB**

Saisir le script et l'exécuter avec la commande Exéc

*Aide pour la saisie du programme :*

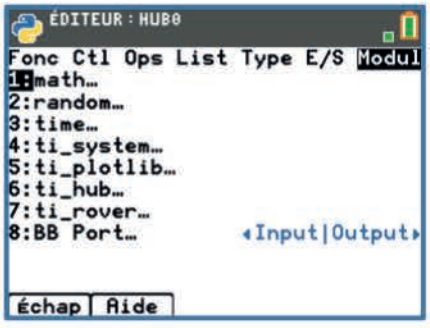
Ligne 1 : **from bbport import ***
Le module **bbport** s'importe avec :

Fns... > Modul **6:ti_hub** puis **3:dispositifs de sortie** et **C:BB Port**

Ligne 2 : **d=bbport(1)**
On créé une instance de **bb_port** pour le port **BB 1** et que l'on nomme **d** :
Aller dans **Fonctions > modules** :

Fns... > Modul
L'écran montre alors de nouvelles fonctions issues du module importé **bbport** :

Faire alors :
**8:BB Port…** **1:var=bb_port(mask)** et adapter en **d=bbport(1)**

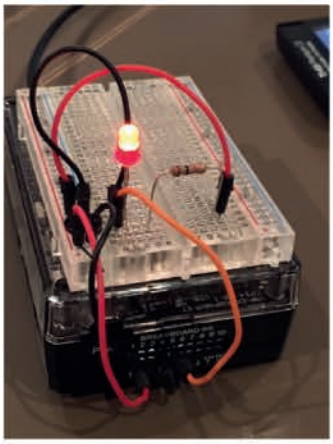



Ligne 3 : `d.write_port(1)`
On met le port numérique BB 1 au niveau haut (5V). Si on voulait mettre ce port au niveau bas (0V), on écrirait : d.write_port(0)
Ici aussi, l'instruction se trouve avec :

Fns... Modul `8:BB Port…` puis
`3:var.write_port(val)`

La diode devrait s'allumer : de manière intense et lumineuse avec la résistance de 1kΩ, et de manière moins intense avec celle 10kΩ.


résistance en série 10kΩ


résistance en série 1kΩ

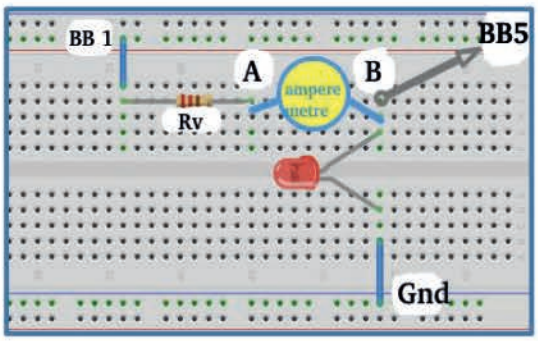
## Proposition de résolution : partie 2

### 1) Modifier le script et le circuit pour réaliser des mesures

- **Pour mesurer l'intensité du courant dans la branche :**

Remplacer le fil entre A et B par un ampèremètre.

Mesurer alors l'intensité du courant dans cette branche et choisir la résistance qui donnera le meilleur éclairement (normalement, celle de 1000).




- **Pour mesurer la tension $U_B$ aux bornes de la DEL :**

On utilisera la calculatrice comme voltmètre. Il faudra alors apporter les modifications au programme précédent, pour lire la tension sur la borne **BB 5**, et afficher la valeur de tension.

Le signal de tension est numérisée sur 14 bits (valeurs comprises entre 0 et $2^{14}$-1, correspondant à des mesures de tension entre 0 et 5V). La variable **lect** stocke cette valeur numérique.

Pour afficher une valeur en Volt sur notre écran, il faudra convertir **lect** et l'affecter à une nouvelle variable, comme par exemple **u** :

$$u = lect \times \frac{5}{2^{14}}$$

**2) Saisie du script : (voir le code complet plus bas dans le document)**

La librairie **analog_in** est un module **de ti_hub**, classé dans la partie dispositifs d'entrée :

Fns... > Modul **6:ti_hub** puis **2:Dispositifs d'entrée** et **8:analog_in**

Le module **analog_in** apporte alors les fonctions qui permettent d'écrire les instructions :

- `i=analog_in("BB 5")`
- `lect=i.measurement( )`

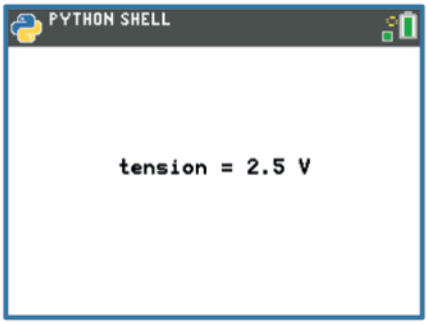
Le module **ti_system** apporte les fonctions pour :

- `while not escape( ) :`
- `sleep(1)`

Le module **ti_plotlib** apporte les fonctions pour :

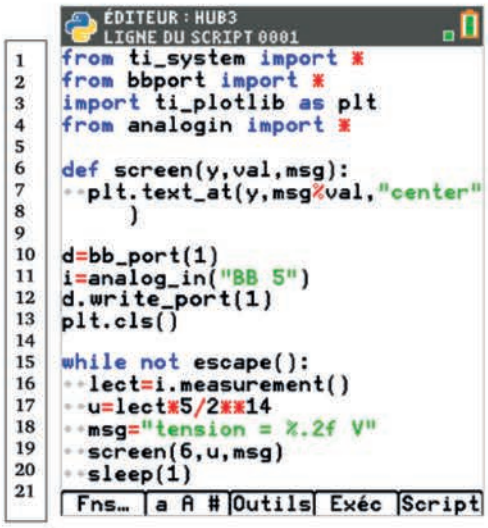
- `plt.cls`
- `plt.text_at(6,msg, "center")`

**Exécuter le script.** L'écran de la calculatrice sert alors d'écran du voltmètre. La tension étant mesurée entre les points BB 5 et l'une des masses de la platine du Hub, brancher la borne BB5 sur le point B du circuit et relever la mesure affiché à l'écran.



**Détails et code du script :**

- Ligne 6-7 : Pour afficher la tension à l'écran : on utilise la même **`fonction screen( )`** que celle définie dans la fiche « ***Ondes et signaux : produire un son, niveau seconde*** ».
- Ligne 10 et 12 : instanciation de l'objet *bb_port pour BB 1* (sortie numerique) et mise au *niveau haut (5V).*
- Ligne 11 : instanciation de l'objet analog_in pour le port BB 5 (entrée analogique).
- Ligne 13 : on efface l'écran.
- Ligne 16 : lecture du port sur BB5.
- Ligne 17 : conversion de la valeur numérique mesurée en tension (V).
- Ligne 18 : La chaine de caractères msg est construite avec un caractère d'échappement, %, qui permet d'afficher la valeur u, de type numérique et avec une précision de 2 décimales après la virgule (.2f).
- Ligne 19 : écriture centrée sur l'écran de la tension mesurée.
- Ligne 20 : temps d'attente pour éviter le clignotement.



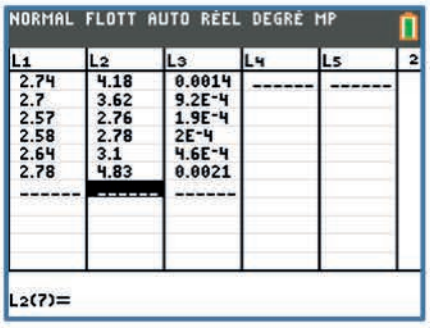

### 3) Relever des mesures

Entre chaque essai, la resistance Rv est changée. Notez sur un support les couples de valeurs mesurées pour $U_B$ et i.

Une fois toutes les mesures réalisées, quitter le programme en appuyant sur la touche ON de la calculatrice.

Pour remplir le tableau de valeurs, comme sur l'image plus bas :

- Quitter le module python et revenir n **mode calculatrice.**
- **Saisir** les valeurs de $U_B$ dans la liste $L_1$
- **Saisir** les valeurs de $U_A$ dans la liste $L_2$
- **Calculer les valeurs de la liste $L_3$ avec : $(L_2-L_1)/1000 \rightarrow L_3$**




Les manipulations utilisant le tableurs de la calculatrice sont expliquées en détail dans la fiche Loi de Descartes, niveau seconde.

*Remarque : Variante sur la manipulation*

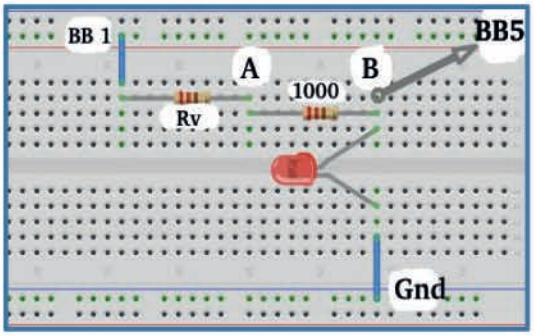
Si vous ne disposez pas d'un ampèremètre susceptible d'être branché sur la plaquette d'essais : Vous pouvez utiliser une variante du circuit précédent, dans lequel vous remplacez le fil AB par une autre resistance de 1kΩ.

La mesure de la tension AB peut se faire à l'aide du fil connecté à BB5 :

- Relever la valeur de la tension $U_B$ sur le point B.
- Puis relever la valeur de la tension $U_A$ sur le point A.
- Calculer l'intensité du courant $i = \frac{U_B - U_A}{1000}$ si la resistance insérée vaut 1kΩ.

### 4) Exploiter les mesures à l'aide d'un langage de programmation

Une fois les valeurs de tension $U_B$ et de courant I, stockées dans les listes L1 et L3, il faudra les lire depuis le module python.
Pour réaliser ceci, on importe depuis l'application python de la calculatrice, les valeurs des listes avec la fonction **`recall_list("numero_de_la_liste").`**



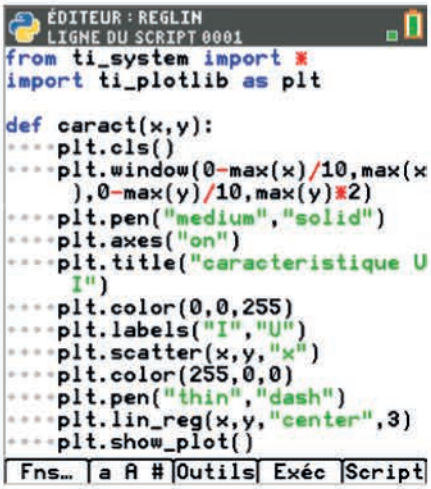

- Créer un nouveau programme python que l'on nommera **RLIN** par exemple (RLIN pour régression linéaire).

- Importer les librairies :
  - ✓ ti_system : celle ci apporte la fonction : **`recall_list( )`**
  - ✓ ti_plotlib : apporte des fonctions utiles pour tracer le graphique et réaliser la modélisation à partir du nuage de points.

- Utiliser les fonctions suivantes de matplotlib :
  - ✓ **`plt.windows(x_min,x_max,y_min,y_max)`** : définit l'échelle pour la fenêtre graphique.
  - ✓ **`plt.scatter(x,y,"type de marqueur")`** : affiche la courbe en mode nuage de points.
  - ✓ **`plt.lin_reg(x,y,1)`** : pour déterminer l'équation de regression linéaire et l'afficher sur l'écran.
  - ✓ **`plt.show_plot( )`** : pour afficher le graphique.

- Saisir le script suivant :

Celui-ci créé une fonction **caract** qui permettra de tracer la courbe y en fonction de x, puis de déterminer son équation de modélisations, et d'afficher la courbe modélisée sur le même écran.

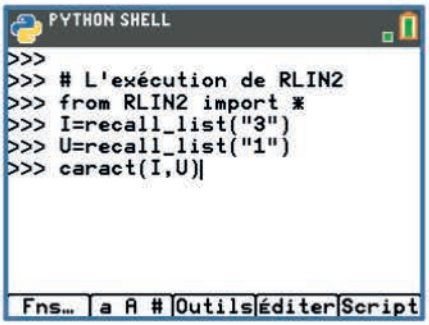



- Exécuter et revenir dans le shell.

- Saisir alors les instructions suivantes :

Ces instructions vont permettre de générer les listes I et U à partir des listes saisies en mode calculatrice.

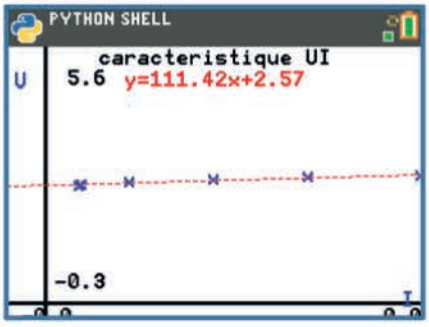
Puis d'appeler la fonction **caract( I,U ).**




On obtient pour la DEL une équation de modélisation :

$$U = 111.4 \times I + 2.57$$

Sa résistance interne est alors de $111\Omega$ et sa tension seuil $2.57V$.




Documents et scripts à télécharger à l'adresse :
https://education.ti.com/fr/physique-chimie

# Fiche méthode

## Référentiel, compétences

**Capacités exigibles :**

- Mesurer une grandeur physique à l'aide d'un capteur électrique résistif.
- Utiliser un dispositif avec microcontrôleur et capteur.

*Compétences numériques mises en œuvre :* notions transversales de programmation.

## Commentaires de l'auteur

Cette séance demande une certaine autonomie et expertise dans l'utilisation du matériel et de la programmation de la calculatrice. Elle doit suivre celle: « *mesure de tensions et de courants. Caractéristique d'un dipôle.* »
Le circuit utilisé, ainsi que le programme python sont des adaptations de ceux utilisés pour cette séance.

## Matériel

- Calculatrice TI-83 Premium CE Edition Python.
- Le microcontrôleur TI-Innovator™ Hub est utilisé avec les connecteurs de la platine (breadboard). On utilisera :
  - ✓ L'une des entrées analogiques en 14 bits (parmi BB5 BB6 ou BB7).
  - ✓ L'une des sorties numériques (parmi les autres BB). Ce sera l'une de ces sorties numériques , par exemple BB1, qui fournira la tension d'alimentation (0V ou 5V).
  - ✓ La sortie de tension fixe 5V (en bas à droite du breadboard.

- Photo résistance type VT900 serie (light dependant resistor LDR).
- Diode électroluminescente DEL d'intensité du **courant nominal 10mA**, plaquette d'essais, fils.
- Conducteurs ohmiques, deux résistances de 1kΩ.
- Dispositif d'éclairage à 5 niveaux d'intensité lumineuse. **(1).**

**(1) dispositif d'éclairage à 5 niveaux d'intensité :** une boite en carton sera perforée avec un gabarit. Chacune des ouvertures sera : **(fig1)**
- soit libre (la première ouverture)
- soit fermée par une à 4 feuille(s) de papier calque.

Le capteur-actionneur électronique sera mis sous l'une des ouvertures.
Une lampe de bureau sera ajustée au-dessus du capteur. La lumière l'atteignant par l'une des ouvertures de la boite en carton. On déplacera l'ensemble (capteur-actionneur) et lampe selon l'ouverture choisie. **(fig2)**

# Thème: signaux et capteurs : capteur de lumière

Niveau: Seconde

TI-83 Premium CE

python

*Eric Tixidor*

## Fiche méthode

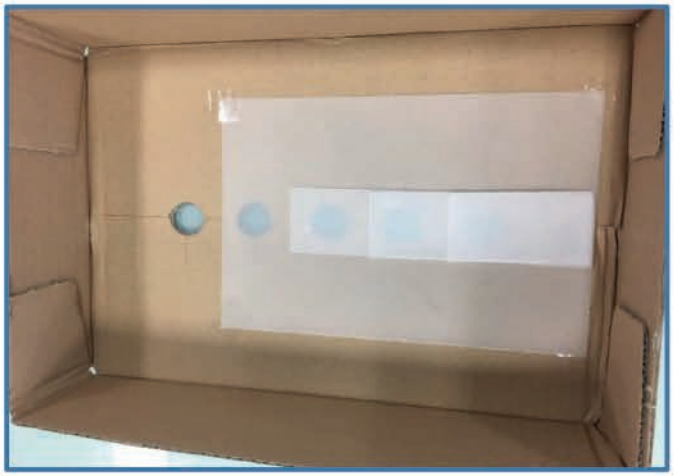


**Fig 1**

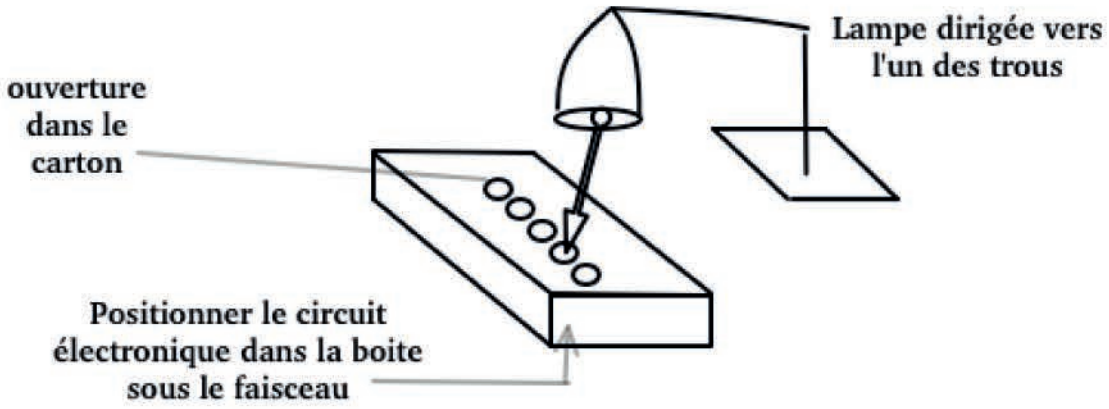



**Fig 2**

## Enoncé

Les capteurs électroniques ont souvent pour rôle de commander ou d'actionner un dispositif en fonction d'une valeur mesurée. Par exemple, commander la vitesse d'un moteur, l'allumage ou l'extinction d'un chauffage, d'un éclairage.
Pour cette séance, vous serez muni d'un dispositif d'éclairage à 5 niveaux d'intensité lumineuse.
Vous allez réaliser un *capteur de lumière (2)* qui allumera une DEL lorsque le circuit se trouve dans *la pénombre*(3)

Les résultats obtenus doivent être *reproductibles*. Les ouvertures fournissent une intensité lumineuse décroissante à mesure que le nombre de feuilles de papier calque superposées dans l'ouverture augmente. Le circuit électronique ne sera éclairé que par la lampe, et protégé de la lumière ambiante par les bords du carton.

**(2) Capteur de lumière :**
Circuit électronique qui transforme le niveau d'éclairement reçue par la photorésistance en une tension.
Cette tension, mesurée directement par le circuit, est comparée à une valeur seuil et déclenchera l'allumage de la DEL.

**(3) La pénombre** correspondra à une lumière qui traverse l'une des ouvertures du cache : La dernière ouverture (celle où l'on a disposé le plus grand nombre de feuilles de papier calque superposées).

## Proposition de résolution

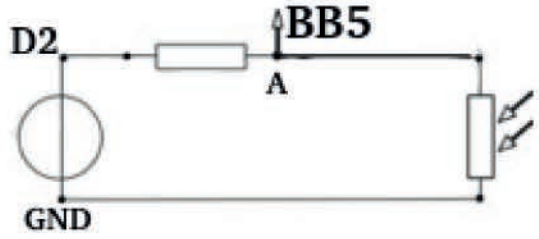
### 1) Réalisation du circuit de mesure d'intensité lumineuse

#### 1.1) Principe

Nous allons réaliser d'abord la partie « détecteur » du capteur de lumière. Ce premier circuit sera de type « diviseur de tension » et utilisera la photorésistance.

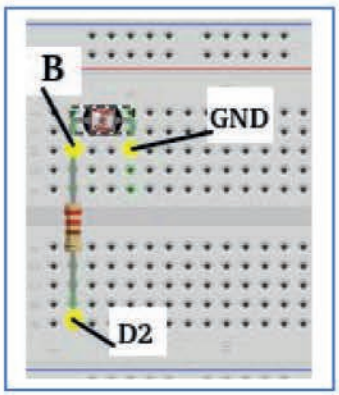
La photorésistance sera mise en série dans une branche comprenant :
- Une alimentation de tension fixe, au point D2. Utiliser l'une des sorties de tension du Le TI-Innovator Hub. Le mieux serait d'utiliser celle de 5V (en bas à droite du breadboard).
- Une résistance fixe de 1kΩ.

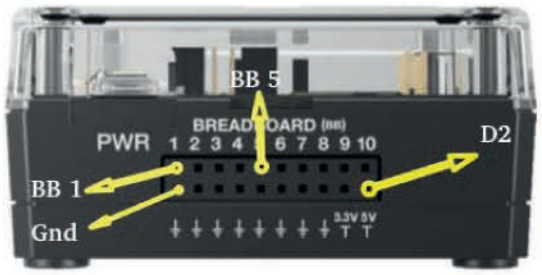
Le point de connexion A entre la résistance et la photorésistance sera relié à l'entrée BB 5 du breadboard pour permettre la mesure de tension.




#### 1.2) Réalisation du circuit

Sur une plaquette d'essai, on relie les composants comme proposé ci-dessous :



## 2) Mesure des intensités lumineuses

### 2.1) Programmer

On adaptera le script réalisé lors de la séance mesure de tensions et de courants (voir fiche précédente) :

Ce programme utilise une boucle non bornée qui est parcourue jusqu'à ce que l'on appuie sur la touche :


Il effectue :

- la mesure du signal **i** sur le port **BB5.**
- la conversion de la valeur numérique **i** en une valeur de tension **u**, comprise entre 0 et 5V.
- l'affichage de la valeur de **u** sur l'écran.

```
from ti_system import *
import ti_plotlib as plt
from analogin import *

def screen(y,val,msg):
    plt.text_at(y,msg%val,
    "center" )

i=analog_in("BB5")
plt.cls()

while not escape():
    lect=i.measurement()
    u=lect*5/2**14
    msg="tension = %.2f V"
    screen(6,u,msg)
    sleep(1)
```

### 2.2) Mesurer

Brancher alors le Hub à la calculatrice. Disposer le circuit sous le faisceau de la lampe, et mesurer l'intensité lumineuse à travers chacune des ouvertures

Exemple de valeurs obtenues :

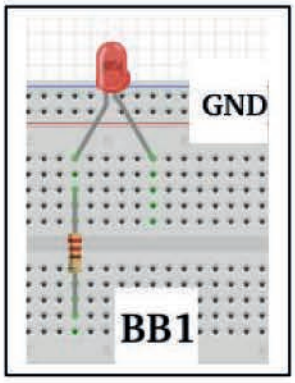
| Nombre de feuilles de papier calques | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Tension $U_A$ (V) Aux bornes de la photorésistance | 0,3 | 1,1 | 2,2 | 3,3 | 4,4 |

On constate que la tension mesurée est une fonction décroissante de l'intensité lumineuse reçue.

## 3) Compléter le circuit avec la branche de la DEL

Sur une autre partie de la plaquette d'essai, réaliser le circuit d'alimentation de la DEL, munie d'une résistance de protection (1kΩ), et alimentée par la sortie numérique **BB1** du breadboard.

Ce circuit est identique à celui qui a été vu lors de la séance *« mesure de tensions et de courants. Caractéristique d'un dipôle. »*



### 4) Programmer le capteur de lumière

**Le programme va permettre :**
- de réaliser les mesures de tension sur BB5.
- d'alimenter la branche avec la DEL avec la sortie BB1.

**Quel travail de programmation peut-on demander aux élèves ?**
Il n'est pas exigé des élèves qu'ils possèdent une expertise particulière en programmation. On peut toutefois leur demander de comprendre le programme en le lisant, d'expliquer son fonctionnement, ou de pouvoir l'adapter.

Ici, on pourra demander par exemple à l'élève d'être capable d'organiser les blocs de programme qui vont :

- permettre d'utiliser une nouvelle sortie numérique, **BB1**
- de définir une nouvelle fonction, appelée **alim**, qui prend en paramètre la valeur **u**, et qui met la sortie de **BB1** :
  - ✓ au niveau 5V lorsque la tension **u** est supérieure à 4
  - ✓ au niveau 0V lorsque la tension **u** est inférieure ou égale 4.
- appeler cette fonction **alim** dans le corps de la boucle principale.

```
from ti_system import *
from bbport import *
import ti_plotlib as plt
from analogin import *
```

```
d1=bb_port(1)
i=analog_in("BB 5")
plt.cls()
```

```
while not escape():
    lect=i.measurement()
    u=lect*5/2**14
    msg="tension = %.2f V"
    screen(6,u,msg)
    sleep(1)
    [          ]
```

```
u > 4 :
```

```
d1.write_port(1)
```

```
d1.write_port(0)
```

```
def  alim(u):
    if [          ]
        [          ]
    else:
        [          ]
```

```
def screen(y,val,msg):
    plt.text_at(y,msg%val,"cente
      r" )
```

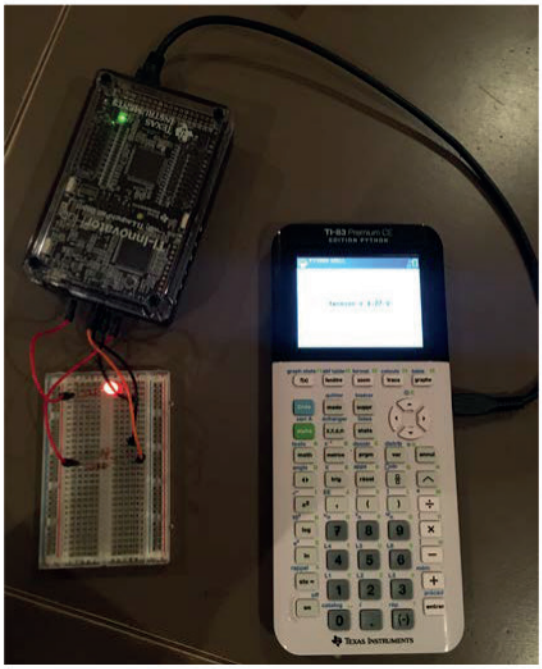
On pourra se référer à la fiche *« mesure de tensions et de courants. Caractéristique d'un dipôle »* pour des explications plus détaillées sur ces instructions.

On rappelle que ce programme doit permettre d'actionner **l'éclairage de la DEL** lorsque **l'intensité** lumineuse reçue est **faible**, c'est à dire, lorsque la tension aux bornes de la photoresistance **u a une valeur élevée**.

Il faudra alors choisir **une valeur seuil** comprise entre **3,3V et 4,4V** d'après le tableau de mesures.

Cela correspondra à actionner l'allumage pour le circuit positionné sur le trou n°5 de la boite. Celui pour lequel on aura mis 4 feuilles de papier calque pour occulter la lumière (pénombre).

On montre ici un exemple de fonctionnement de ce capteur de lumière pour 2 conditions d'éclairage :



Circuit dans la pénombre : LED allumée

Circuit sous éclairage : LED éteinte

## Correction : Script HUB complet

ÉDITEUR : HUB3
LIGNE DU SCRIPT 0001

```
from ti_system import *
from bbport import *
import ti_plotlib as plt
from analogin import *

def screen(y,val,msg):
    plt.text_at(y,msg%val,"center"
        )
def alim(u):
    if u>4:
        d1.write_port(1)
    else:
        d1.write_port(0)

d1=bb_port(1)
i=analog_in("BB 5")
plt.cls()

while not escape():
    lect=i.measurement()
    u=lect*5/2**14
    msg="tension = %.2f V"
    screen(6,u,msg)
    sleep(1)
    alim(u)
```

Documents et script à télécharger à l'adresse :
https://education.ti.com/fr/physique-chimie

# Thème: transformation de la matière :

**Suivi et modélisation de l'évolution d'un système chimique**

Niveaux: Seconde et 1ere spécialité physique chimie

TI-83 Premium CE

python

*Eric Tixidor*

## Fiche méthode

## Référentiel, compétences

**Capacités numérique :** Déterminer la composition de l'état final d'un système chimique, siège d'une transformation chimique totale à l'aide d'un langage de programmation.

## Commentaires de l'auteur

Les élèves doivent être déjà familiarisés avec l'environnement python de la calculatrice. De plus, ils doivent déjà connaître, en langage python :

- Les types simples, dont les booléens.
- Les structures de données complexes : listes et dictionnaires.
- Les fonctions.
- Les boucles bornées (parcours dans un dictionnaire) et non bornées (while).
- Les conditions.

Pour la partie chimie : on suppose que la définition de l'avancement chimique est connue.

## Matériel

- Calculatrice TI-83 Premium CE Edition Python.
- Un ordinateur muni d'un IDE python quelconque pour saisir le programme (facultatif).

## Enoncé

On réalisera un programme qui permettra de déterminer l'état final d'un système chimique.
Ce programme doit pouvoir s'adapter à toute transformation chimique dont on connaît l'équation, et les quantités de matière initiales (en mol) pour les réactifs et les produits *(les conditions initiales).*

Pour renseigner l'équation et les conditions initiales on utilisera une structure de données de type dictionnaire.
Soit une réaction chimique d'équation :

$$\mathbf{coef_1\ R_1 + coef_2\ R_2 \rightarrow coef_1\ P_1 + coef_2\ P_2}$$

Si les réactifs *Ri* sont en quantité de matière initiale **`nri`** , et les produits *Pi* en quantité de matière **`npi`** : Le format de données sera le suivant :

```
eq = {"reac":{"R1" : [coef1, np1],"R2" : [coef2,nr2]},
      "prod":{"P1" : [coef1, np1],"P2" : [coef2,np2]}}
```

Le programme devra contenir les fonctions nécessaires au calcul de l'avancement maximum **`x`**, et modifier les valeurs **`nri`** et **`npi`** du dictionnaire **`eq`**.

# Proposition de résolution

## Numériser l'équation et le système chimique

Dans l'éditeur python de la calculatrice : créer un nouveau script que l'on appellera : **AVAN**

On commence par initialiser le dictionnaire **eq** à partir de ses clés "**reac**" et "**prod**". Dans le script **AVAN**, saisir :

```
eq = {"reac":{}, "prod":{}}
```

Pour utiliser les fonctions du script, il faudra commencer par remplir les valeurs pour les clés "**reac**" et "**prod**".

Prenons un exemple. Soit l'équation de la réaction de synthèse de l'eau :

$$H_2 + \frac{1}{2} O_2 \rightarrow H_2O$$

Les coefficients associés à chacune des espèces chimiques sont :
- 1 pour $H_2$
- 0.5 pour $O_2$
- 1 pour $H_2O$.

Supposons les quantités de matière, dans l'état initial égales à
- 2 mol pour $H_2$
- 2 mol pour $O_2$
- 0 mol pour $H_2O$.

Pour représenter cette équation de réaction et le système chimique dans l'état initial, on devra alors écrire :

⇨ Pour les réactifs :

```
eq["reac"] = {"H2": [1,2], "O2": [0.5,2]}
```

⇨ Pour le(s) produit(s) :

```
eq["prod"] = {"H2O": [1,0]}
```

La valeur associée à la clé "**reac**" est-elle même un dictionnaire, où chacune des clés est la référence à l'une des espèces chimiques, parmi les réactifs. Ces clés sont uniques.

La valeur associée à chaque clé "*H2*" et "*O2*" est une liste comprenant 2 valeurs :
- La première valeur, au **rang 0** de la liste, contient la valeur numérique du **coefficient stœchiométrique** (ici, on mettra 1 pour "*H2*" et 0.5 pour "*O2*".
- La deuxième valeur, au **rang 1**, contient le **nombre de mole** de l'espèce dans l'état initial.

Pour le produit de réaction, à la clé "*H2O*", le coefficient stœchiométrique étant égal à 1, et sa quantité de matière à l'état initial sera nulle. On met alors [1,0] pour valeur.
Une autre possibilité serait de renseigner directement le contenu du dictionnaire ***eq*** avec l'instruction unique suivante :

```
eq = {"reac":{"H2": [1,2], "O2":
[0.5,2]}, "prod":{"H2O": [1,0]}
}
```

Le tableau d'avancement à l'état initial sera le suivant :

| equation | $H_2$ | + | 0,5 $O_2$ | = | $H_2O$ |
|---|---|---|---|---|---|
| avancement | `reac["H2"][1]` | | `reac["O2"][1]` | | `prod["H2O"][1]` |
| x = 0 mol | 2 | | 2 | | 0 |

## Fonction `fin`

La fonction **fin** détermine si le système est à **l'état final** (alors elle retourne **`True`**) ou s'il peut encore « avancer » (retourne **`False`**).
Cette fonction prend un paramètre, **reac**. C'est une variable qui devra contenir la valeur associée à la clé "**reac**" du dictionnaire ***eq***.

Dans le script, on saisira :

```
def fin(reac):
  b=True
  if reac!={}:
    b=False
    for i in reac.keys():
      if reac[i][1]<=0:
        b = True
  return b
```

Le programme parcourt tous les éléments (toutes les clés) du dictionnaire **reac**, quel que soit le nombre d'espèces chimiques, à condition qu'une espèce chimique au moins ait été renseignée.

A chaque itération (pour chacun des réactifs), la valeur du nombre de mol est comparée à 0. Si cette valeur est inférieure ou égale à zéro, la fonction renvoie True (pour marquer la fin de la réaction chimique).

## Fonction `avance`

La fonction **`avance`** prend en paramètres :

- La variable **`eq`**, qui contient les paramètres de l'équation chimique et de l'état du système chimique.
- Une valeur **`dx`** qui représente **le pas** avec lequel on fera croitre l'avancement **`x`** à chaque itération.

Cette fonction va modifier la valeur des quantités de matière pour chacune des espèces (valeur des listes "**reac**" et "**prod**" au rang 1) et retourne l'avancement **x** lorsque le premier réactif arrive à zéro.

A la suite du programme, saisir la fonction suivante :

```
def avance(eq,dx):
  x=0
  reac = eq["reac"]
  prod = eq["prod"]
  while fin(reac)==False: # il
reste des reactifs
    x+=dx
    for i in reac.keys():
      reac[i][1]-=reac[i][0]*dx
    for j in prod.keys():
      prod[j][1]+=prod[j][0]*dx
  return x
```

Tant qu'il reste des réactifs (c'est-à-dire **while fin(reac)==False** ), alors :

- L'avancement est augmenté de **dx** avec : **x+=dx**
- Pour les réactifs, la quantité de matière est diminuée de **dx***coefficient_stœchiométrique :

```
reac[i][1]-=reac[i][0]*dx
```

- Pour chaque produit, la quantité de matière est augmentée de dx*coefficient_stœchiométrique :

```
prod[j][1]+=prod[j][0]*dx
```

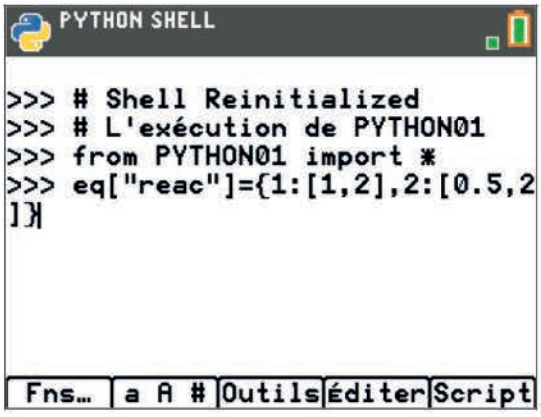
La fonction retourne finalement la valeur de l'avancement *maximum*.

```
  return x
```

## Utiliser le programme

Exécuter le programme. Puis, dans le shell, écrire les instructions pour numériser le système chimique et son équation. Pour une écriture plus rapide, on écrire une clé numérique à la place des formules chimiques des espèces. L'exemple précédent devient alors :

⇨ Pour les réactifs :



⇨ Puis pour le produit :

```
>>> eq["prod"]={1:[1,0]}
```

Appeler alors la fonction **avance** avec pour paramètres, la variable **eq**, et le pas égal à **0.1**

```
>>> avance(eq,0.1)
2.0
```

La fonction retourne alors la valeur de l'avancement maximum, pour le système chimique à l'état final : **2.0** mol.

On peut également consulter l'état final du système, en tapant **eq** :

```
>>> eq
{'reac': {2: [0.5, 1.0], 1: [1,
-6.38378239159465e-16]}, 'prod'
 {1: [1, 2.0]}}
```

Ce qui peut être représenté par le tableau d'avancement suivant :

| equation | $H_2$ | + | 0,5 $O_2$ | = | $H_2O$ |
|---|---|---|---|---|---|
| avancement | reac["H2"][1] | | reac["O2"][1] | | prod["H2O"][1] |
| x = 0 mol | 2 | | 2 | | 0 |
| x mol | 2-x | | 2-0,5*x | | x |
| x = 2.0 mol | 0.0 | | 1.0 | | 2.0 |

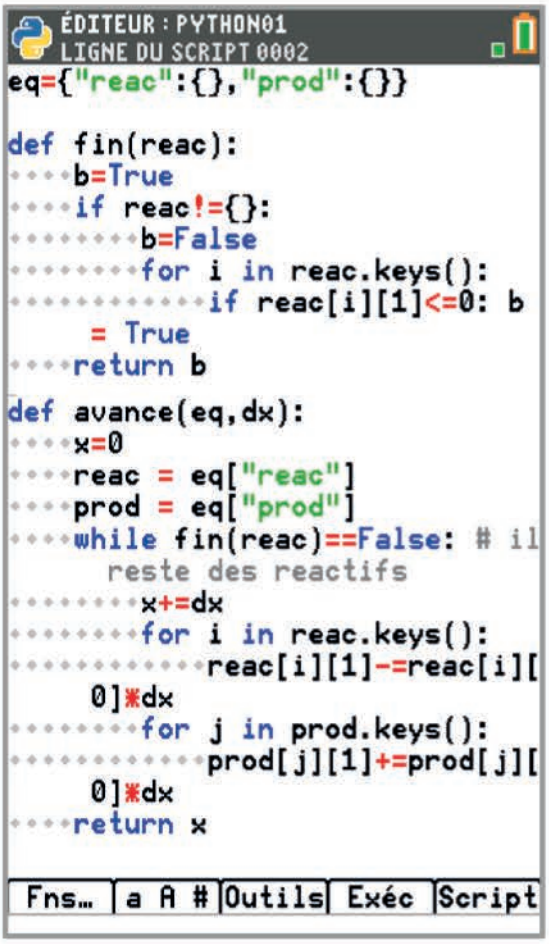
## Ecriture du programme dans un autre editeur

Le programme complet est relativement court, mais si l'on souhaite y ajouter des commentaires, il peut être plus pratique d'utiliser un IDE python pour le saisir, puis l'envoyer vers la calculatrice à l'aide du logiciel TI Connect CE.

Pour réaliser cette manipulation :

- Enregistrer le programme une fois celui-ci saisi dans l'ordinateur. Par exemple avec le nom *avancement.py.*
- Connecter la calculatrice à l'ordinateur. Lancer TI Connect CE.
- Appuyer sur le bouton  (*Ajouter les données de l'ordinateur à la ou aux calculatrices connectées)*
- Débrancher alors la calculatrice et lancer son application python. Chercher alors le programme qui vient d'être distribué à la calculatrice. Le nom a certainement été modifié lors du transfert ; il a peut-être été changé en PYTHON01

## Script **AVAN** complet




Documents et script à télécharger à l'adresse :
https://education.ti.com/fr/physique-chimie

# Thème: Désintégration radioactive :

**Evolution du nombre de noyaux et demi-vie**

Niveau: Première enseignement scientifique

TI-83 Premium CE

python™

*Eric Tixidor*

## Fiche méthode

## Référentiel, compétences

**Capacités exigibles :**

- Utiliser une représentation graphique pour déterminer une demi-vie.

**Capacités numériques :**

- Faire *un usage explicité des outils numériques*.

## Commentaires de l'auteur

Les capacités numériques mises en jeu ici sont *non exigibles*. Mais cette séance permettra de consolider les notions de programmation en *langage python* étudiées dans d'autres matières de la classe de première.

Le BO suggère également de « **Mettre en œuvre des pratiques scientifiques » :** *modéliser, simuler, raisonner …* C'est cette liberté qui rend possible la réalisation de cette activité en classe.

## Materiel

- Calculatrice TI-83 Premium CE Edition Python.

## Prérequis

Les élèves doivent être déjà familiarisés avec l'environnement python de la calculatrice.

## Enoncé

**Le sujet :** Dans ce travail pratique, on utilisera un script en Python pour simuler l'évolution de la population du carbone 14 au cours du temps.

Les nucléides cosmogéniques, comme le $C^{14}$ peuvent être formés dans l'atmosphère mais aussi directement dans les minéraux des roches après impact des rayons cosmiques.

Leur population radioactive décroit par radioactivité et suit la loi de Poisson :

$$\Delta N = -\lambda \Delta t \times N$$

avec $N$ le nombre de noyaux.

$\lambda$ est la constante radioactive du radionucléide : Cette valeur représente la *probabilité de désintégration* d'un noyau par unité de temps. Pour $C^{14}$ on a $\lambda = 0{,}000121\ an^{-1}$.

## Fiche méthode

Cette loi exprime que la variation (diminution) **ΔN** du nombre **N** de noyaux pendant un temps court **Δt** dépend du produit :

- de la constante radioactive $\lambda$
- du temps **Δt**
- et du nombre **N** de noyaux restants.

Le temps **Δt** doit être choisi suffisamment court, pour que le produit **λ.Δt** soit inférieur à 0.1, voire, du même ordre de grandeur.

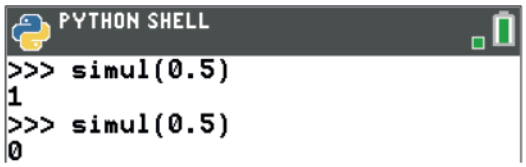
On pourra choisir, par exemple, un intervalle **Δt = 1000**. (soit 1000 ans). Ce qui donnera :

$$\lambda \times \Delta t = 0{,}000121 \times 1000 = 0{,}121$$

Cette valeur se situe alors dans la limite supérieure de validité pour le traitement suivant. Les résultats donnés par ce modèle sont toutefois assez satisfaisants. Il ne faudra pas prendre de valeur plus élevée pour **Δt.**

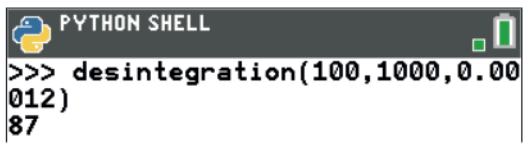
**Travail à réaliser :**

1) Ecrire une fonction **`simul`** qui prend comme paramètre $p$ et qui revoie 1 avec une probabilité $p$ et 0 avec une probabilité $1-p$.

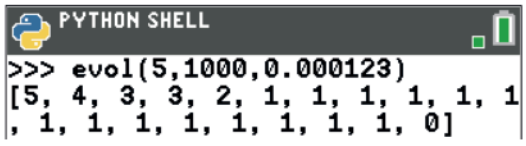



2) Ecrire une fonction **`desintegration`** qui prend comme paramètres $N$ le nombre de noyaux (initial), $dt$ et $l$ (pour $\lambda$) et qui renvoie le nombre de noyaux restants après le temps $dt$.

Pour un temps de $dt = 1000$ la probabilité de désintégration est $p = \lambda \times dt$.




3) Ecrire une fonction **`evol`** qui prend comme paramètres $N$ le nombre de noyaux (initial), $dt$ et $l$ (pour $\lambda$) et qui renvoie une liste contenant le nombre de noyaux restant à chaque pas $dt$ (la liste doit commencer par $N$ et se terminer par 0).




4) Utiliser les outils graphiques de la calculatrice pour déterminer le temps de demi-vie du carbone $C^{14}$.

# Proposition de résolution

## Fonction `simul`

1) Par définition, la probabilité de désintégration est :

$$\mathbf{p} = \lambda \times \Delta \mathbf{t}$$

On peut proposer deux façons de programmer cette fonction :

- Une première très classique qui choisit aléatoirement un réel dans l'intervalle $[0,1]$ et qui renvoie 1 si ce nombre est plus petit que la valeur de probabilité **p** et 0 sinon.

ÉDITEUR : RADIO
LIGNE DU SCRIPT 0008

```
from random import *
def simul(p):
    a=random()
    if a<=p:
        return 1
    else:
        return 0
```

- Une seconde qui consiste à choisir aléatoirement un réel dans l'intervalle $[0,1]$ et de lui ajouter **p**. Ce qui revient à choisir un réel **a** dans $[p; 1+p]$. Parmi ces nombres, ceux qui sont dans $[1,1+p]$ ont une partie entière égale à 1. Et il y a une probabilité **p** que le nombre **a** soit dans cette intervalle.

ÉDITEUR : RADIO
LIGNE DU SCRIPT 0006

```
from random import *
def simul(p):
    a=random()+p
    b=int(a)
    return b
```

Dans les deux cas, il faudra importer la librairie random à l'aide de l'instruction :

**from random import** $*$

## Fonction `desintegration`

2) Pour chaque particule on va déterminer si elle se désintègre ou non en appelant **`simul(dt*l)`.**

Si le résultat est nul, alors la particule ne se désintègre pas, si le résultat vaut 1 alors elle se désintègre.

Il suffit donc de retirer ce résultat à N pour obtenir le nouveau nombre de noyaux restants.

On recommence ce travail pour chaque noyau dans une boucle **`for`.**

ÉDITEUR : RADIO
LIGNE DU SCRIPT 0016

```
def desintegration(N,dt,l):
    for i in range(N):
        N=N-simul(dt*l)
    return N
```

## Fonction `evol`

3) Pour obtenir la liste contenant le nombre de noyaux restant à chaque pas $dt$, on l'initialise avec une première valeur N : **`liste=[N]`.**

Puis, tant qu'il reste des noyaux (c'est-à-dire **`while N>0`**) on calcule le nombre de noyaux restant après **dt** et on l'ajoute à la **`liste`.**

ÉDITEUR : RADIO
LIGNE DU SCRIPT 0021

```
def evol(N,dt,l):
    liste=[N]
    while N>0:
        N=desintegration(N,dt,l)
        liste.append(N)
    return liste
```

## Détermination graphique du temps de demi vie

4) Ajouter au début du script la librairie ti_system :

```
from ti_system import *
```

Executer le script. Cela aura pour effet de charger les fonctions et de les rendre accessible depuis le *shell.*
Dans le *shell* : lancer la fonction avec les paramètres suivants :

- **N** : nombre de noyaux à l'instant initial : 100
- **dt** : le pas du temps discrétisé : 1000 ans
- **l** : constante radioactive : 0.000121 $an^{-1}$.

La fonction doit retourner la liste du nombre de radionucléides, comptés pour chaque intervalle de durée **dt**.
On devra stocker ces données dans une nouvelle liste, accessible depuis le shell, par exemple **no**.
L'instruction s'écrira alors :

```
no = eval(100,1000,0.000121)
```

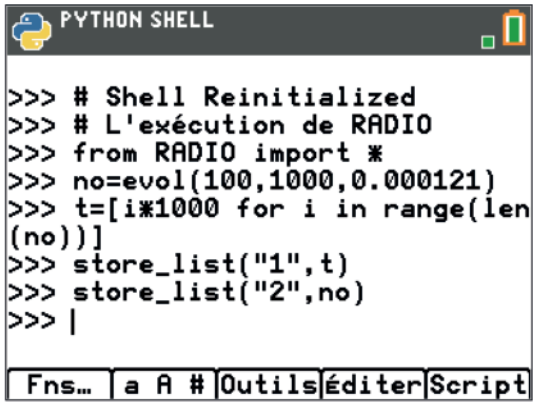
Créer alors une liste avec les repères de temps correspondants. Cette liste, appelons la **t**, aura la même longueur que **no**. Et sera remplie avec des multiples de **dt**, ce qui donne (avec **dt=1000**) :

```
t = [i*1000 for i in
range(len(no))]
```

On suppose pour la suite que les listes $L_1$ et $L_2$ du menu STAT de la calculatrice ont été au préalable effacée.

Stocker **t** et **no** dans ces deux listes grâce à la fonction **store_list** de la librairie **ti_system** :

- **store_list("1",t)**
- **store_list("2",no)**




Sortir alors de l'application python.

# Thème: Désintégration radioactive :

## Evolution du nombre de noyaux et demi-vie

Renseigner les listes à prendre pour les axes du graphiques $L_1$ et $L_2$ :

Dans le menu graph stat, accessible depuis [2nde] [f(x)] (graph stats f1) **1:Graph1…Aff,** choisir :

Afficher la courbe : **Aff**

XListe : remplacer la liste par L1 à l'aide de la combinaison de touches [2nde] [stats] (listes) **1:$L_1$**

YListe : choisir **$L_2$**

Vérifier les valeurs des extrema des deux listes en parcourant les valeurs de celles-ci :

[stats] (listes) **1:modifier**

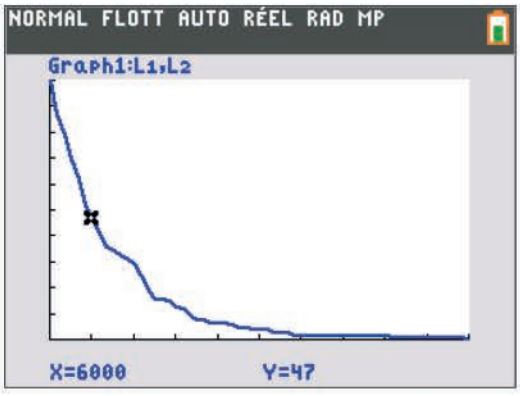
La valeur maximale pour l'axe des Y est connu : il s'agit de la valeur de **N** à l'instant initial (**no**=100).
La dernière valeur de **t** donnera la valeur maximale à renseigner pour l'axe des X. Probablement 30000 avec les conditions initiales utilisés.

Modifier les axes : [fenêtre] (déf table f2), puis modifier les valeurs pour afficher au mieux la courbe à l'écran.

Puis afficher le graphique : [graphe] (table f5) ainsi que le curseur sur la courbe : [trace] (calculs f4)

Déterminer alors (graphiquement) la valeur du temps de demi-vie du carbone $C^{14}$ à partir de cette simulation :

Vous devriez trouver une valeur comprise entre 5000 et 6000 ans.



## Modification des paramètres

Les détails de la courbe obtenue montrent que la simulation est assez imprécise. La courbe est loin d'être lisse et présente des discontinuités. De plus, l'intervalle de temps qui encadre la durée de demi-vie est assez important (intervalle de 1000 ans). On améliorera la précision de cette valeur, et la qualité de la courbe en prenant d'autres paramètres pour la fonction **`eval(N,dt,l).`** Essayer par exemple :

- ✓ **`N`** : 400
- ✓ **`dt`** : 200
- ✓ **`l`** : 0.000121

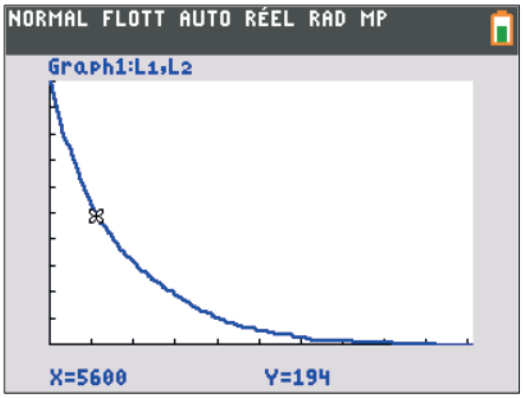
Revenir dans l'application python. Exécuter le script RADIO. Dans le shell, adapter les instructions vues précédemment pour calculer les noyaux avec ces nouveaux paramètres.
Stocker les valeurs de **`t`** et **`no`** dans les listes $L_1$ et $L_2$ de la calculatrice.

Quitter l'application python, puis :

- adapter les axes X et Y de la fenêtre graphique à partir des nouvelles valeurs obtenue
- effacer le graphique précédent pour permettre l'affichage de la nouvelle courbe à l'aide du menu DESSIN :


Le nouveau tracé doit alors apparaître à l'écran.

Le temps de demi-vie devrait avoir une valeur plus proche de celle attendue et la courbe plus lissée (valeur theorique : ***5750 ans***).




## Prolongement

On pourra adapter ce script au calcul du temps de demi-vie pour d'autre radionucléides : il suffira d'adapter la valeur de la constante radioactive, l, par rapport au tableau suivant :

Et choisir une valeur de **`dt`** adaptée.

| nucléides | constante radioactive ($an^{-1}$) |
|---|---|
| $H^3$ | 0.0565 |
| $Be^{10}$ | $4.6*10^7$ |
| $Cl^{36}$ | $2.3*10^6$ |

## Script RADIO

ÉDITEUR : RADIO
LIGNE DU SCRIPT 0003

```
from random import *
from ti_system import *

def simul(p):
  a=random()+p
  b=int(a)
  return b

def desintegration(N,dt,l):
  for i in range(N):
    N=N-simul(dt*l)
  return N

def evol(N,dt,l):
  liste=[N]
  while N>0:
    N=desintegration(N,dt,l)
    liste.append(N)
  return liste
```

Fns… | a A # | Outils | Exéc | Script

Documents et script à télécharger à l'adresse :
https://education.ti.com/fr/physique-chimie

## Plus d’informations sur le site internet de T³ France :

- Des ressources pédagogiques pour votre classe
- Un programme de formations gratuites sur site et en ligne
- Des vidéos d’aide à la prise en main de la technologie

t3france.fr

Un service après-vente est également accessible depuis le site **education.ti.com/fr/csc**