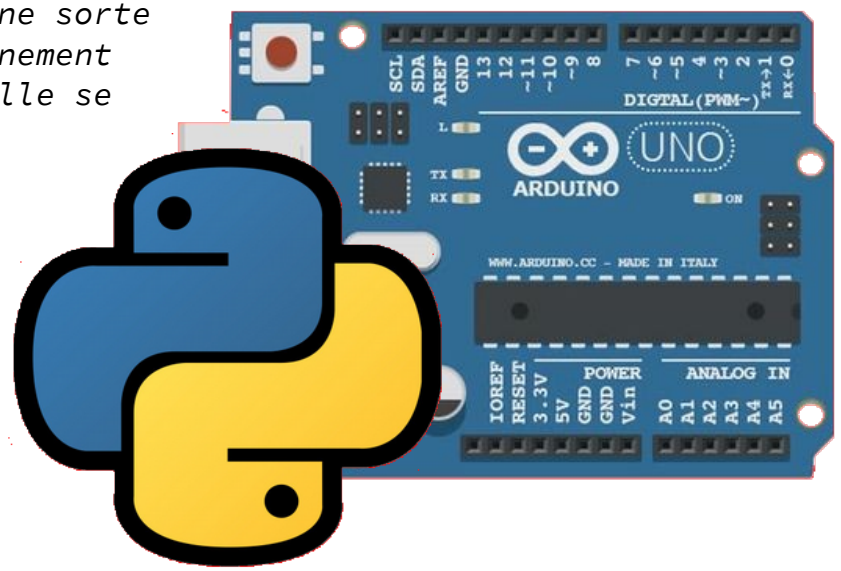
Nouveaux programmes du lycée
Physique-chimie

Utilisation de la carte Arduino UNO en langage Python

La carte Arduino est un microcontrôleur, c'est à dire une sorte de mini ordinateur qui sert d'interface entre l'environnement (actions, mesures de grandeurs...) et un utilisateur. Elle se programme nativement en langage C.

Le langage Python est un langage de programmation libre et gratuit, utilisé pour les calculs scientifiques.

*Ce document présentera l'utilisation de la carte **Arduino** en langage **Python**. Cela permettra ainsi aux élèves de n'avoir à utiliser qu'un seul et même langage pour une partie des nouveaux programmes du lycée.*

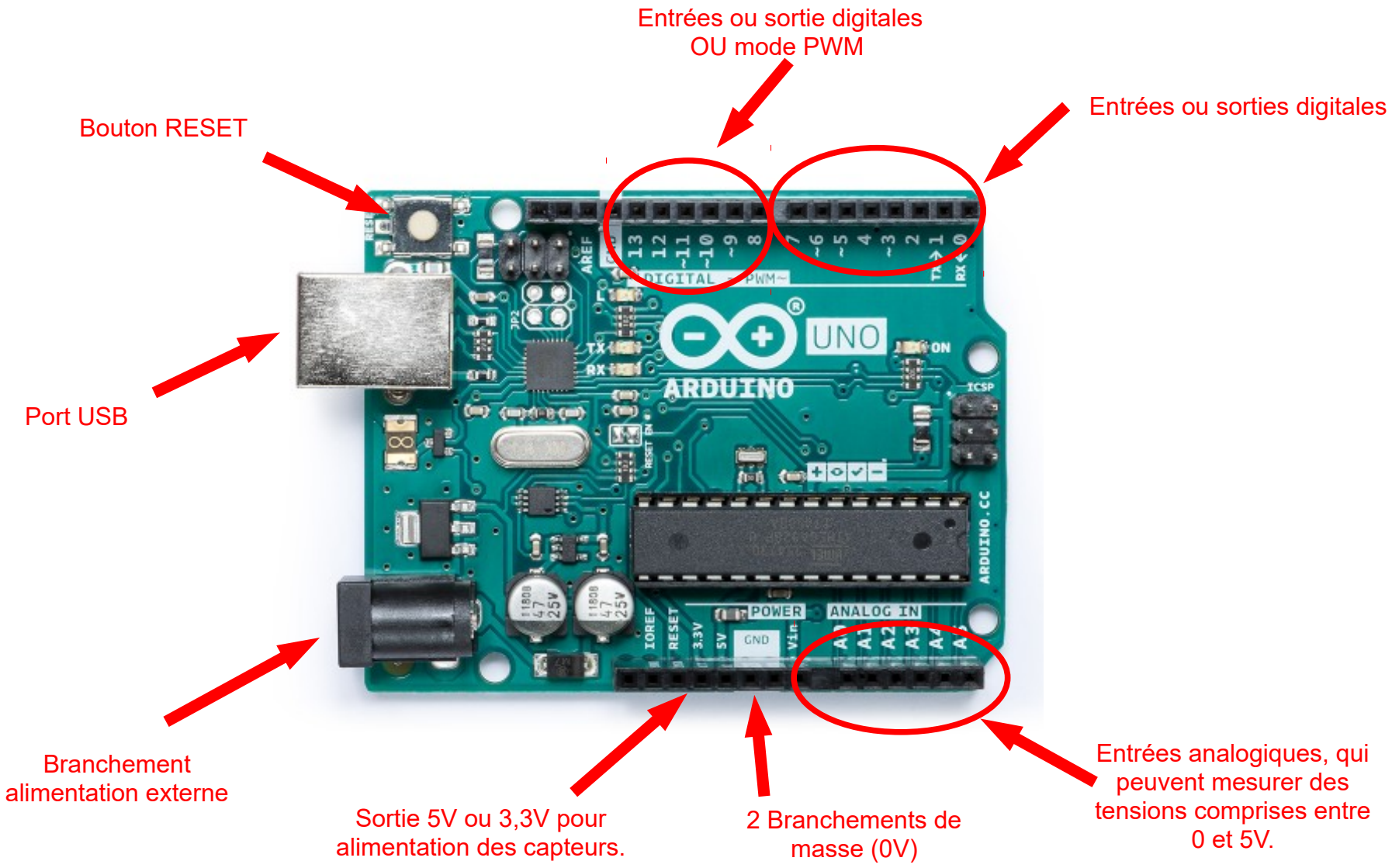


AVERTISSEMENT : Cette configuration ne permet pas une utilisation à fréquence d'échantillonnage élevée de la carte Arduino : les manipulations type « générer un signal sonore » ou « mesurer une distance par ultrasons » ne sont donc pas possibles.

Sommaire

Matériel à prévoir :
- Une carte Arduino Uno (originale ou copie)
- un ordinateur
- le logiciel IDE Arduino, un logiciel IDE Python (Thonny, Edupython, Pyzo, Spyder ...)
- quelques composants (une LED, une Résistance CTN ou sonde PT100, différentes résistances…)

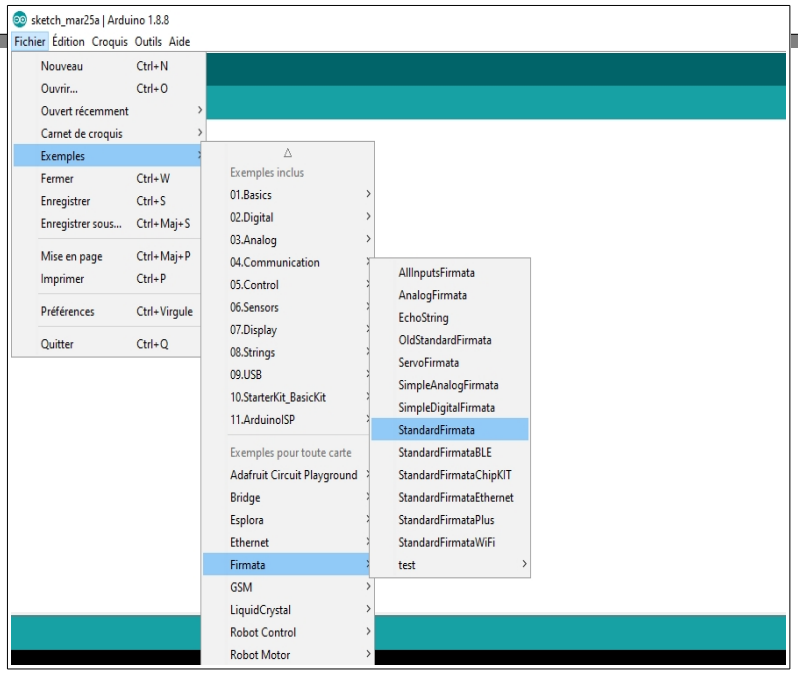
Les différentes parties de la carte



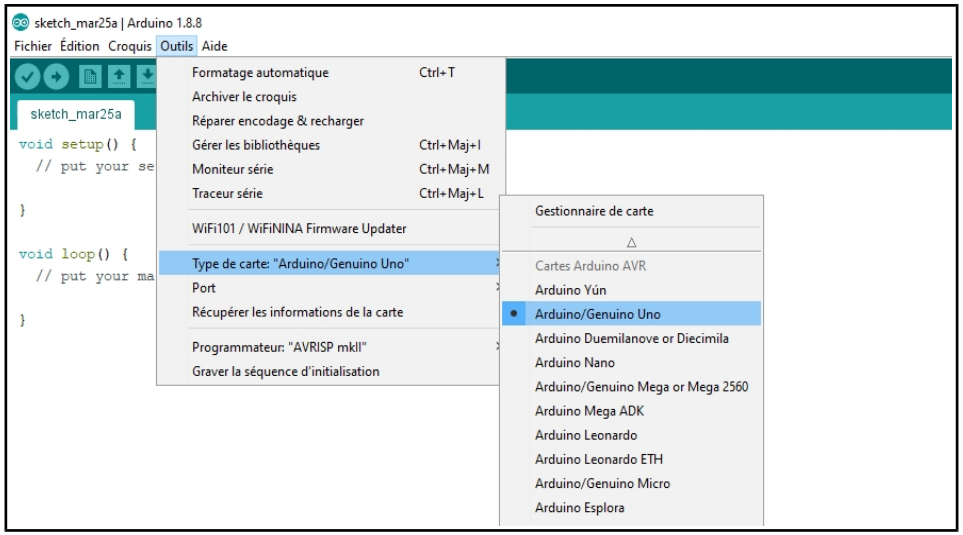
Étape 1 :Installation du Firmware sur la carte

La première étape consiste à installer un firmware différent sur la carte pour qu'elle puisse communiquer en **Python**. Cette étape peut-être réalisée par un adulte avant la séance :

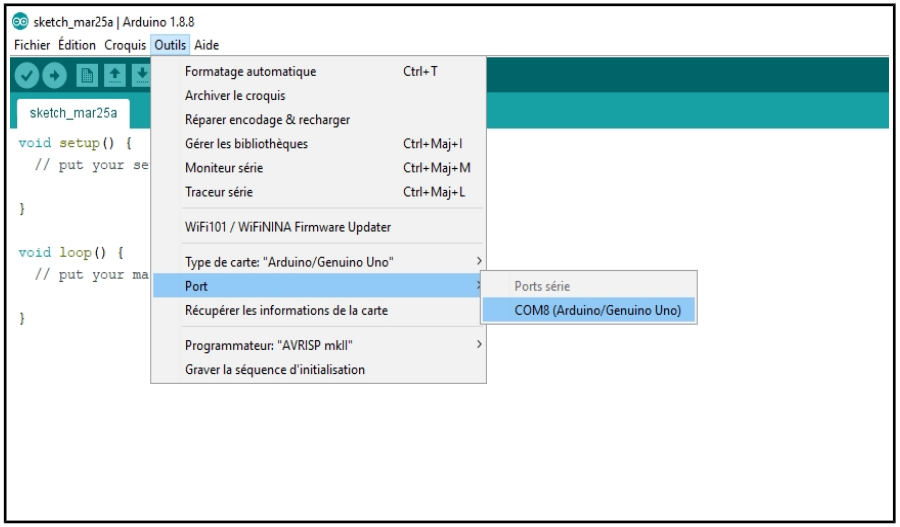
Installer et ouvrir le logiciel Arduino.
Puis **fichier > Exemples > Firmata > StandardFirmata**



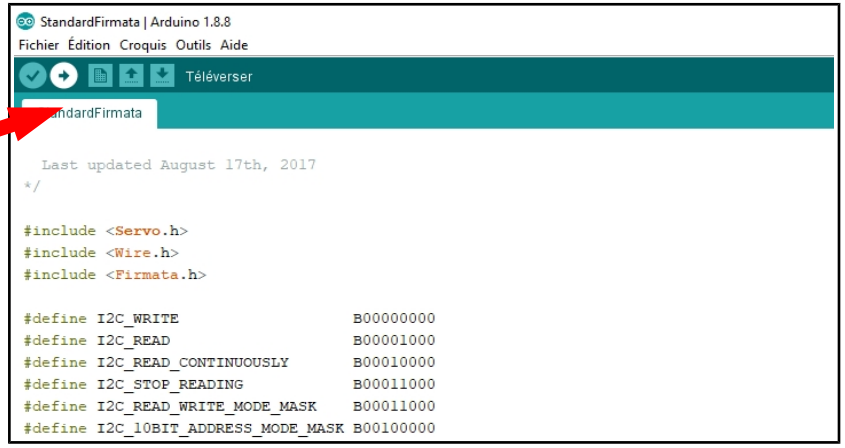
Brancher la carte Arduino Uno en USB. Choisir le type de carte **Outils > Type de carte >Arduino Uno**



Puis connecter la carte :
Outils > Port > COMX
(la carte connectée apparaît dans la liste)



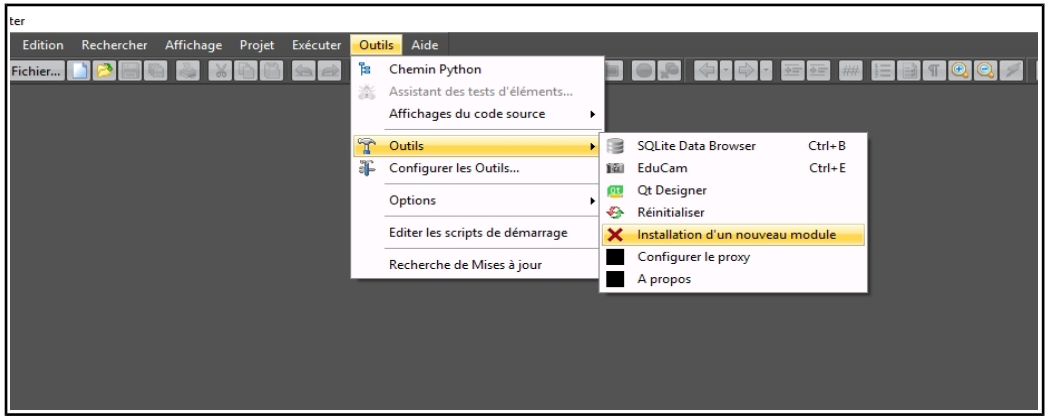
Il ne reste plus qu'a téléverser le microprogramme (firmware) sur la carte :
Icône ⇨



La carte est prête pour être utilisée avec un IDE Python comme **EduPython**. Ce firmware reste ensuite sur la carte le temps de l'activité, même si elle est déconnectée ou éteinte.

Étape 2 : Téléchargement de la bibliothèque Pyfirmata dans Edupython

Pour utiliser la carte avec **EduPython**, il faut installer la bibliothèque **Pyfirmata**, qui renferme les commandes **Python** compréhensibles par la carte **Arduino**.



Démarrer EduPython.
Faire **Outils > installation d'un nouveau module.**

Un menu apparaît
Choisir **2** et ensuite taper **pyfirmata**.

Suivre les instructions.

Une procédure similaire existe pour Thonny, Pyzo, Spyder….

Étape 3 : les principales fonctions de pyfirmata

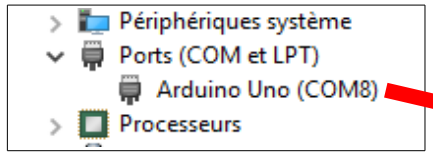
Nous allons pouvoir commander la carte **Arduino** grâce au logiciel **EduPython**.

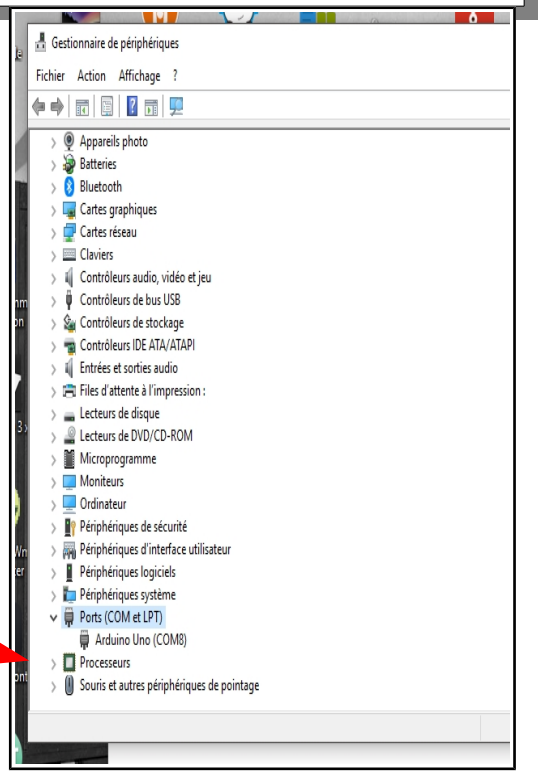
- Démarrer **EduPython**.

- Brancher la carte **Arduino** Uno.

En cas d'oubli, pour retrouver le port de communication (COM) de la carte, pour cela, il faut ouvrir le **gestionnaire de périphériques** dans le menu **Démarrer** de Windows.

La carte reliée à l'ordinateur apparaît dans la liste, avec le numéro de port. Ici c'est **COM8**.





Dans **Edupython**, Nous pouvons taper un code qui sera reconnu par la carte, à condition de le faire commencer par les lignes suivantes :

Importation du module pyfirmata

Choisir un nom pour la carte(« carte »)
et son adresse (« COM8 »)

Ces deux lignes lancent la fonction mesure en temps réel

```
from pyfirmata import Arduino, util
import time

carte = Arduino('COM8')
acquisition = util.Iterator(carte)
acquisition.start()
```

Les entrées et sorties sur la carte se définissent grâce à la fonction **get_pin()**
Avec pour un choix de paramètres dans la parenthèse :

a	analogique	Le numéro de branchement sur la carte (de 0 à 13)	**i**	input , entrée
d	digitale		**o**	output, sortie
p	pwm (non traité ici).			

Par exemple :

entree1 est l'entrée analogique A0
sortie1 est une sortie digitale en 13.

La ligne **time.sleep(1.0)** génère une pause de 1s indispensable pour l'initialisation de la carte.

```
from pyfirmata import Arduino, util
import time

carte = Arduino('COM8')
acquisition = util.Iterator(carte)
acquisition.start()

entree1 = carte.get_pin('a:0:i')
sortie1 = carte.get_pin('d:13:o')
time.sleep(1.0)
```

Remarques importantes :
*- le module **time** doit être importé en début de code.*
- les noms de variables « carte », « acquisition », « entree1 », « sortie1 » ont été choisis par le rédacteur du code, il est donc possible de les nommer comme on le souhaite (si possible de manière explicite).
*- La fonction **carte.exit()** à la fin du code, termine l'acquisition de mesures proprement.*

Nous allons pouvoir commencer les mesures et actions avec les fonctions **read()** et **write(*valeur*)**
Plusieurs cas sont possibles :

Entrée		**Sortie**
Analogique	**Digitale**	**Digitale**
read()	**read()**	**write(*valeur*)**
Lis la valeur de tension sur l'entrée analogique. Cette fonction renvoie un nombre type **Float** compris entre 0 et 1 qui correspond à une tension comprise entre 0V et 5V	Lis l'état de la tension sur l'entrée analogique. (type **Boolean**) **0** pour 0V (False) **1** pour 5V (True)	Impose une tension sur la sortie : 0V pour write(**0**) 5V pour write(**1**). *Il est possible d'utiliser* ***write(****False****)*** *ou* ***write(****True****)***

Code test

Ce petit code rapide permet de tester (sans aucun matériel) si tout est opérationnel. Il consiste à faire clignoter 10x la LED qui se trouve d'origine sur la carte et qui est reliée à la sortie digitale 13.

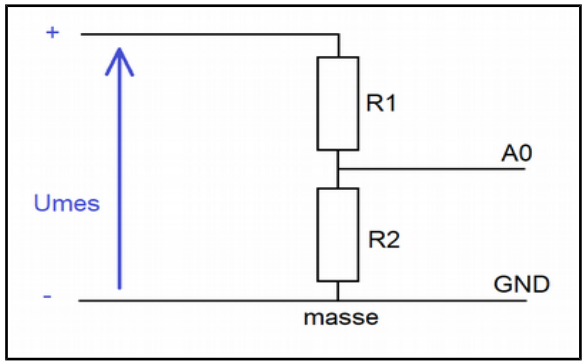
```
from pyfirmata import Arduino, util
import time

carte = Arduino('COM8')
acquisition = util.Iterator(carte)
acquisition.start()
led13 = carte.get_pin('d:13:o')
time.sleep(1.0)

print("Début du test")
for i in range(0,10):
    led13.write(1)
    time.sleep(0.5)
    led13.write(0)
    time.sleep(0.5)
print("Fin du test")
carte.exit()
```

Montage 1 : Mesurer d'une tension électrique

Les entrées analogiques ne pouvant mesurer que des tensions de valeur maximale 5V, il faut utiliser deux résistances en pont diviseur de tensions, pour créer un voltmètre mesurant une tension supérieure.

La valeur de la tension **Umes** est donnée par la formule :

$$U_{mes} = 5\,V\,.\,tension_{A0}\,.\,\frac{R_1+R_2}{R_2}$$

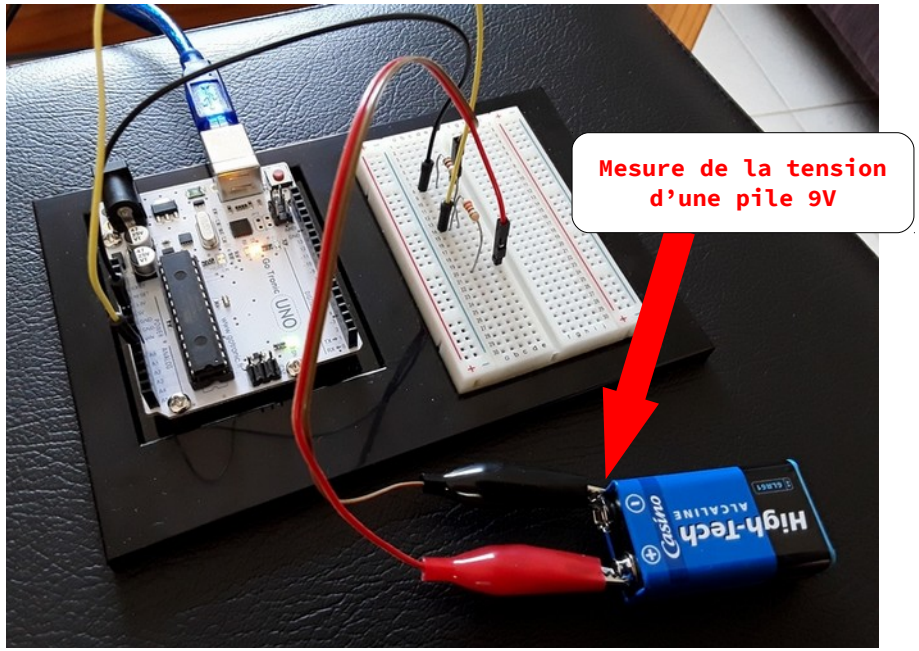


(tension_A0 est une valeur type FLOAT comprise entre 0 et 1, mesurée sur l'entrée analogique A0. Il faut donc multiplier par 5V pour obtenir la vraie valeur)

Par exemple :

Si on prend R1 = 3000Ω et R2 = 1000Ω, U_{mes} max sera égale à 5V*1*(3000+1000)/1000 = 20V.

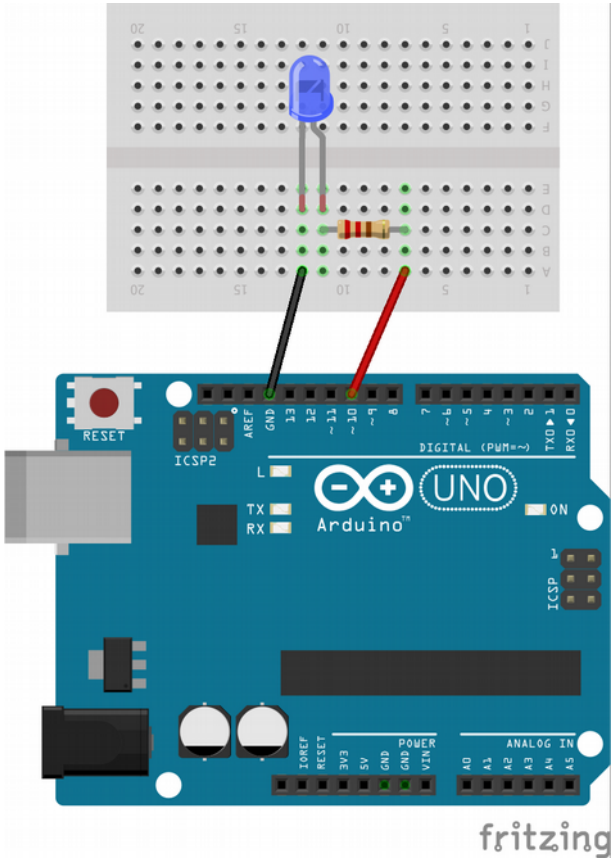
```
from pyfirmata import Arduino, util
import time

carte = Arduino('COM8')
acquisition = util.Iterator(carte)
acquisition.start()

tension_A0 = carte.get_pin('a:0:i')  # entrée A0
time.sleep(1.0)    # temps d'initialisation de la carte

R1 = 3000 #Ohm
R2 = 1000 #Ohm

Umes = tension_A0.read()*5*(R1+R2)/R2
print(Umes, "V")

carte.exit() # clôture les mesures
```



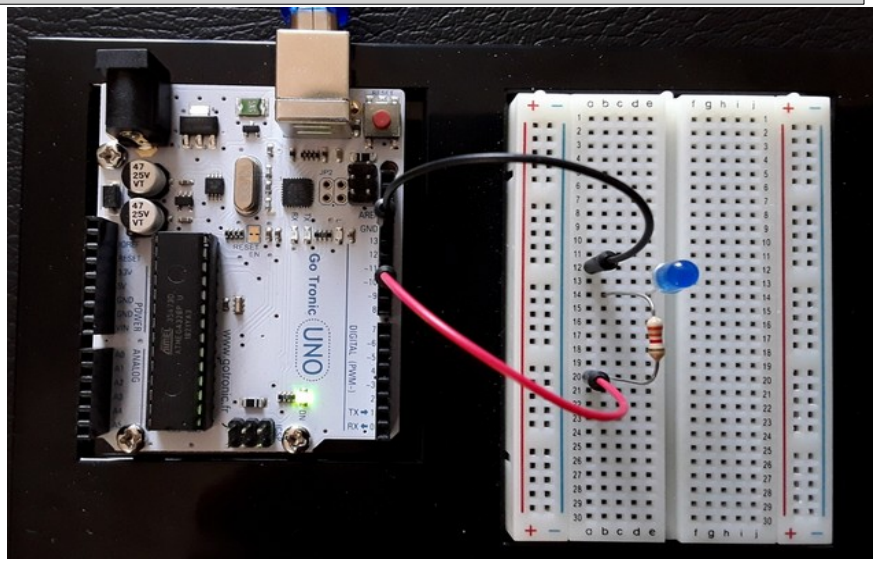
Remarque : ici le programme n'effectue qu'une seule mesure. On pourrait ajouter une boucle pour faire une mesure chaque seconde pendant 1 min. Il faudra alors utiliser ***time.sleep(durée)*** *pour échantillonner la série de mesures.*

Montage 2 : commander une LED :

Voici un petit programme simple pour commander une LED. On pourra par la suite remplacer la LED par un autre composant (buzzer, relais) suivant les besoins, le principe restant le même.



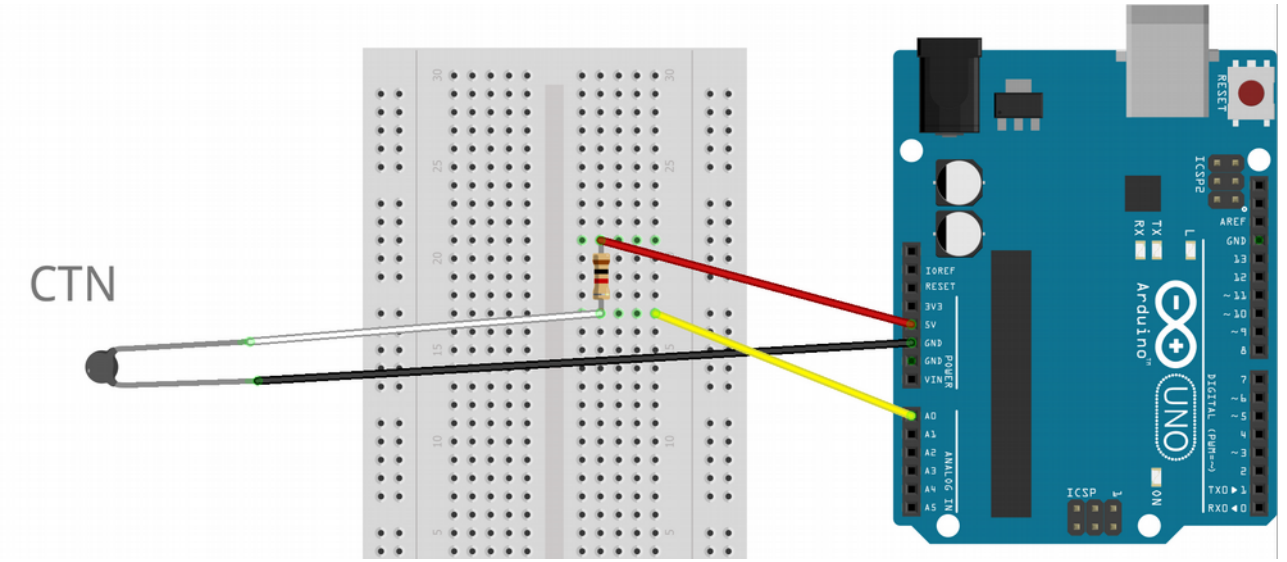

```
from pyfirmata import Arduino, util
import time

carte = Arduino('COM8')
acquisition = util.Iterator(carte)
acquisition.start()

sortie = carte.get_pin('d:10:o')  # voie 10 en sortie digitale
time.sleep(1.0)                   # temps d'initialisation de la carte

sortie.write(True)  # envoie 5V sur la sortie
time.sleep(4)       # attendre 4 secondes
sortie.write(False) # mettre la sortie à 0V
carte.exit()
```

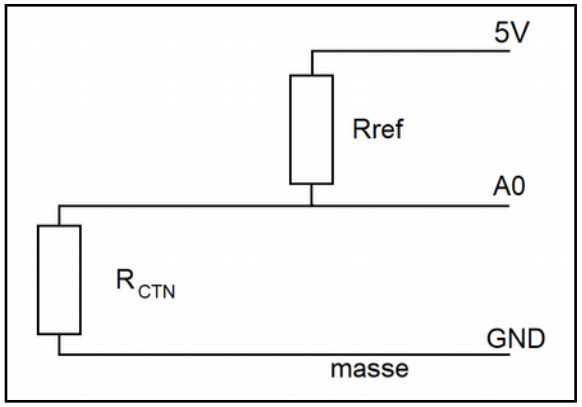


Remarque : L'allumage d'une LED peut être utilisé pour coder une information, par exemple quand une valeur maximale mesurée est atteinte.
Il faudra alors utiliser les fonctions ***if …*** *ou* ***while …***

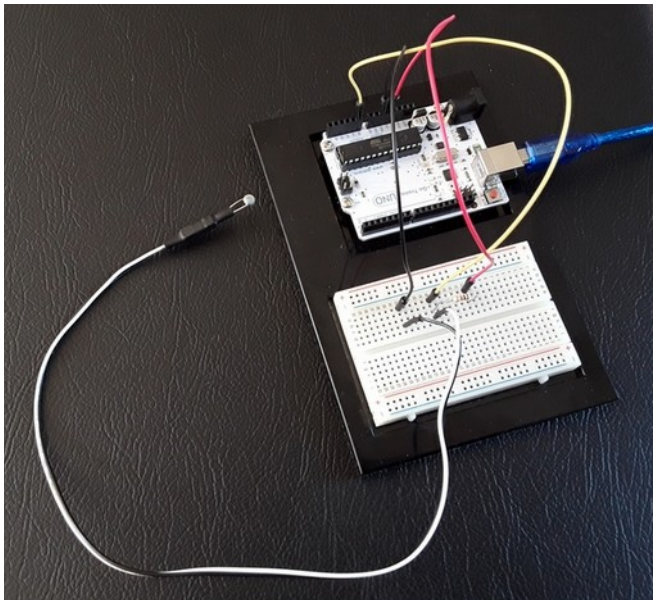
Montage 3 : Mesurer la résistance d'un capteur résistif

La carte **Arduino** permet de déterminer la résistance en **Ω** d'un capteur résistif (sonde de température PT100, photorésistance …).
Ici nous prendrons l'exemple d'une résistance CTN (1kΩ à 25°C).
La résistance doit être placée dans un pont diviseur, ce pont étant alimenté par la sortie 5V de la carte :





R_{ref} = 1000 Ω
Résistance CTN type 1kΩ à 25°C



La valeur de R_{ctn} est donnée par la formule : $R_{CTN} = R_{ref} \cdot \frac{tension_{A0}}{1 - tension_{A0}}$

(**tension_A0** est la valeur renvoyée par la mesure de l'entrée analogique **A0**)

Voici le code qui permet de déterminer la valeur de la résistance CTN :

En mesurant la température en même temps à l'aide d'un thermomètre, et en reportant les valeurs (température, résistance CTN) dans un tableur on obtient facilement une courbe d'étalonnage.

- 60 mesures seront effectuées
- La valeur mesurée s'affiche dans la console
- Attente de 5s entre chaque mesure pour avoir le temps de noter dans le tableur

```
from pyfirmata import Arduino, util
import time

carte = Arduino('COM8')
acquisition = util.Iterator(carte)
acquisition.start()

tension_A0 = carte.get_pin('a:0:i')
time.sleep(1.0)

Rref = 1000 #Ohm

for i in range(0,60):
      tensionCTN = tension_A0.read()
      Rctn = Rref*tensionCTN/(1-tensionCTN)
      print(Rctn)
      time.sleep(5)

carte.exit() # permet de stopper les mesures.
```

Petite astuce : on peut placer la résistance dans un ballon de baudruche pour la rendre étanche, puis la plonger mélange glace/sel (-12°C) et placer ce mélange sur un réchaud en fonctionnement jusqu'à l'ébullition.

La courbe d'étalonnage peut ainsi être tracée et exploitée classiquement avec un tableur ou avec **Python** (voir page suivante)

Exploiter la courbe d'étalonnage avec Python : Sonde linéaire PT 100

Voici une proposition d'exploitation de courbe d'étalonnage pour sonde PT100. Nous avons mesuré ici la résistance aux bornes d'une sonde PT100 dont la résistance varie linéairement avec la température.
Le code suivant permet de tracer le graphique avec matplotlib et de faire une régression en utilisant la fonction **numpy.polyfit** :

Les valeurs de T et R mesurées sont notées dans les listes entre []

Calcul des paramètres **coef[0]** et **coef[1]** tels que T = **coef[0]***R + **coef[1]**

Tracé de la courbe expérimentale
Tracé de la courbe de tendance linéaire

Cette partie définit les titres et les graduations des axes

```
import matplotlib.pyplot as plt
import numpy as np

T = [257.15, 273.15, 293.15, 295.15, 303.15, 315.15,
323.15, 333.15, 343.15, 350.15, 353.15, 363.15, 368.15,
371.15]
R = [96.0, 100.0, 107.0, 109.0, 111.0, 115.0, 119.0,
123.0, 126.0, 129.0, 130.0, 134.0, 134.0, 135.0]

coef = np.polyfit(R, T, 1)

plt.plot(R,T,'b+')

plt.plot([0,150],[coef[1], coef[1]+coef[0]*150], "r")

plt.title("Caractéristique PT100 : T = f(R)")
plt.xlabel('R en Ohm')
plt.ylabel('T en K')
plt.axis([0,150,0,400])

plt.show()
```

Exploiter la courbe d'étalonnage avec Python : Résistance CTN :

Voici l'exploitation de courbe d'étalonnage pour une résistance CTN dont la résistance varie logarithmiquement en fonction de la température.
Il est possible d'utiliser numpy.polyfit() en remplaçant R par ln(R), soit **np.log(R)** en python.
*ATTENTION np.log() est la fonction logarithme népérien venant du module **numpy** de Python.*
Les valeurs de T et R mesurées sont notées dans les listes entre []

Calcul des paramètres **coef[0]** et **coef[1]** tels que T = **coef[0]***ln(R) + **coef[1]**

Calcul de valeurs pour le modèle (courbe de régression).

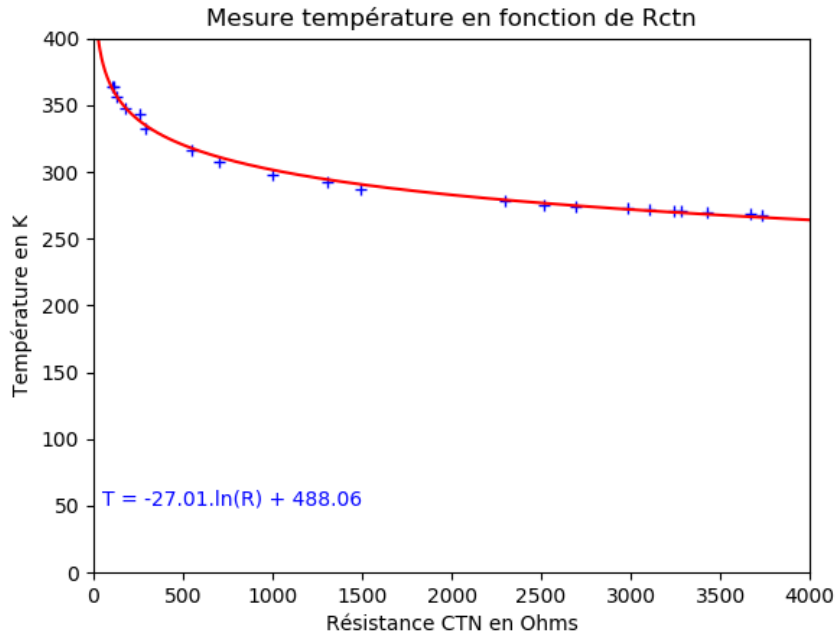
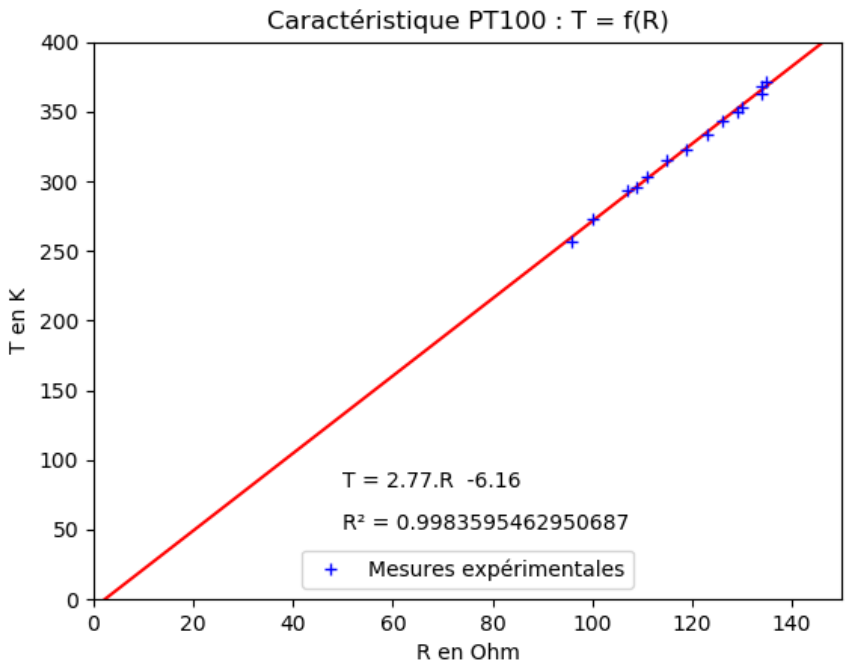
Tracé de la courbe expérimentale

Tracé de la courbe de régression

Cette partie définit les titres et les graduations des axes

```
import matplotlib.pyplot as plt
import numpy as np

T = [268, 269, 270, 271, 271, 272, 273,
274, 275, 278, 287, 292, 298, 308, 316, 333, 343, 348,
356, 364, 364]
R = [3736, 3671, 3430, 3280 ,3244, 3108, 2980, 2693, 2515,
2300 ,1489, 1305,
1000, 705, 545, 288, 258, 180 ,127 ,103, 111]

coef = np.polyfit(np.log(R), T, 1)


T1, R1 = [], []
for i in range(0,4000):
    R1.append(i)
    T1.append(coef[1]+coef[0]*np.log(i))


plt.plot(R,T, "b+")
plt.plot(R1, T1, "r")

plt.axis([0,4000, 0, 400])
plt.title('Mesure température en fonction de Rctn')
plt.xlabel('Résistance CTN en Ohms')
plt.ylabel('Température en K')

plt.show()
```

Voici les deux types de courbes obtenues avec ces deux types de capteurs :

Résistance CTN 1kΩ à 25°C	Sonde type PT100
	

*Remarque : en ajoutant la ligne de code **print(coef[0], coef[1])** au code précédent, Python affichera les valeurs des paramètres pour retrouver l'équation de la courbe d'étalonnage.*

```
*** Console de processus distant Réinitialisée ***
>>>
-27.06749784900115 488.52582604129066
>>>
```

T = -27,067*ln(R) + 488,52

```
*** Console de processus distant Réinitialisée ***
>>>
2.7748056994818664 -6.162564766839569
```

T = 2,77*R - 6.16

Montage 4 : Mesurer la température en utilisant la capteur étalonné

Le montage précédent nous permet d'obtenir la courbe d'étalonnage « température en fonction de la résistance mesurée » et son équation. Il est donc possible de réinjecter cette équation dans un programme Python pour cette fois mesurer directement des températures.

Il faut importer le module **math** pour utiliser la fonction logarithme népérien

*ATTENTION math.log() est la fonction logarithme népérien venant du module **math** de Python.(la fonction np.log vue précédemment marche également)*

La température est calculée ici

Les mesures s'afficheront dans la console :

```
from pyfirmata import Arduino, util
import time
import math

carte = Arduino('COM8')
acquisition = util.Iterator(carte)
acquisition.start()

tension_A0 = carte.get_pin('a:0:i')
time.sleep(1.0)

Rref = 1000 #Ohm
for i in range(0,60):
      tensionCTN = tension_A0.read()
      Rctn = Rref*tensionCTN/(1-tensionCTN)
      temperature = 488-273.15 - 27.1*math.log(Rctn)
      print(temperature, "°C")
      time.sleep(1)

carte.exit()
```

Montage 5 : Afficher un graphique en temps réel avec Matplotlib :

Le montage 4 (mesure de température par CTN) va être réutilisé, mais ce code pourra être adapté à un autre type de mesure.

Matplotlib est une bibliothèque de Python permettant d'afficher des graphiques, nous allons l'utiliser pour afficher les mesures effectuées par la carte en temps réel.

L'initialisation de la carte

Création des 2 listes pour les 2 séries de mesures

Déclaration de la durée de la série de mesure.
Déclaration de l'intervalle de mesure.

Les mesures de temps se feront avec la fonction **time.time()**, qui renvoie un nombre de seconde par rapport à une référence de l'ordinateur : il faut donc mesurer un temps initial et le soustraire.

Mesure de la tension CTN par la carte et calcul de la température.

Ajout des mesures dans les listes.

Définition des axes du graphique [xmin,xmax,ymin,ymax]
Affichage du graphique « r+ » pour des croix rouges. Ne rien mettre pour une courbe en ligne continue.
Le temps d'attente entre chaque mesure.

```
from pyfirmata import Arduino, util
import matplotlib.pyplot as plt
import time
import math

carte = Arduino('COM8')
acquisition = util.Iterator(carte)
acquisition.start()
tension_A5 = carte.get_pin('a:5:i')
time.sleep(1.0)

T, t = [],[]
durée_mesure = 120 #en seconde
interval = 2       #en seconde
nb_mesure = int(durée_mesure/interval)
Rref = 1000
ti=time.time()
for i in range(0,nb_mesure):
    plt.cla()
    tensionCTN = tension_A5.read()
    Rctn = Rref*tensionCTN/(1-tensionCTN)
    temperature = 488-273.15 - 27.1*math.log(Rctn)
    T.append(temperature)
    t.append(time.time()-ti)
    plt.axis([0,durée_mesure,0,50])
    plt.plot(t,T, "r+")
    plt.pause(0.01)
    time.sleep(interval)
plt.show()
carte.exit()
```

Remarque importante : l'utilisation d'Arduino avec Python montre sa limite ici : les mesures de temps sont imprécises et ne doivent pas être trop rapprochées : ne pas choisir un intervalle inférieur à 1s. C'est pour cela que la mesure de distance par capteur ultrasonore reste impossible avec Pyfirmata et Arduino.

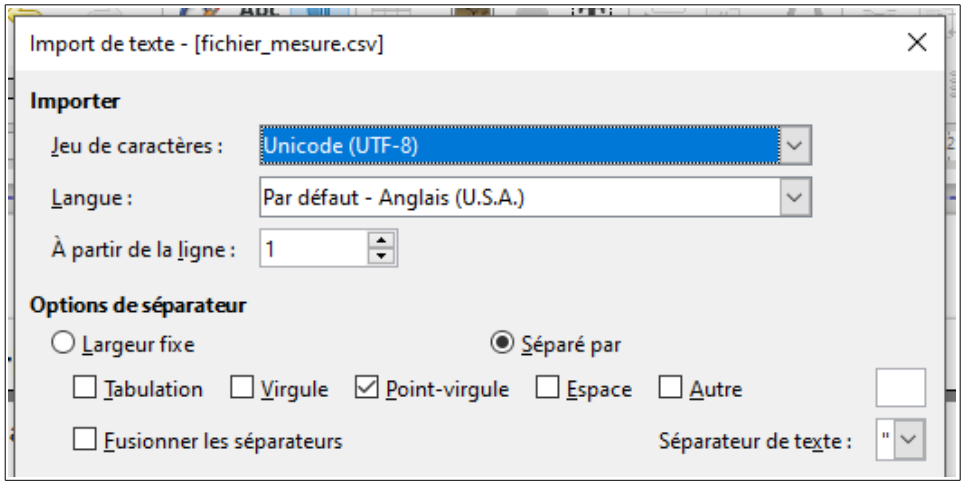
Montage 6 : Exporter une série de mesures au format csv :

Le module csv de Python permet de d'acquérir ou d'exporter des données au format csv. En ajoutant ce morceau de code à la fin du programme, le fichier csv « **fichier_mesure.csv** » est crée dans le même dossier que le programme Python.
Dans cet exemple, deux colonnes sont crées, chacune représentant une série de mesures : ici **t** et **T** du montage 5.

```
…
import csv

…
… # le reste du programme ici
…
with open('fichier_mesure.csv','w', newline = '') as csvfile:
    for i in range(len(t)):
        writer = csv.writer(csvfile, delimiter=';')
        writer.writerow([t[i],T[i]])
```

Ce fichier **fichier_mesure.csv** peut ensuite être ouvert avec un tableur. Choisir « ; ».

Remarque : Ne pas oublier d'importer le module csv en début de code. Les virgules décimales sont notées par des points.



	A	B
1	0.390591144561768	21.2907846998682
2	2.79906415939331	21.2907846998682
3	5.00356292724609	21.4005607419073
4	7.20794296264648	21.5102855428119
5	9.4118869304657	21.5102855428119
6	11.6159961223602	21.5102855428119
7	13.8822820186615	21.6199600372145
8	16.0557501316071	21.6199600372145
9	18.2125151157379	21.6199600372145
10	20.4004600048065	21.6199600372145
11	22.5728371143341	21.7186248405719
12	24.7452809810638	21.7186248405719
13	26.9327731132507	21.7186248405719
14	29.1368219852448	21.7186248405719
15	31.3247420787811	21.7186248405719
16		

Sources - liens

Le site officiel **Arduino** : https://www.arduino.cc/

La documentation officielle **pyfirmata** :https://pyfirmata.readthedocs.io/en/latest/index.html

La physique avec Arduino : https://opentp.fr/card/

La physique autrement :

http://hebergement.u-psud.fr/supraconductivite/projet/enseigner_la_physique_avec_arduino/

Fritzing, le logiciel libre et gratuit pour schématiser les montages : http://fritzing.org/home/