4 Exemples d'utilisation de la carte Arduino en physique

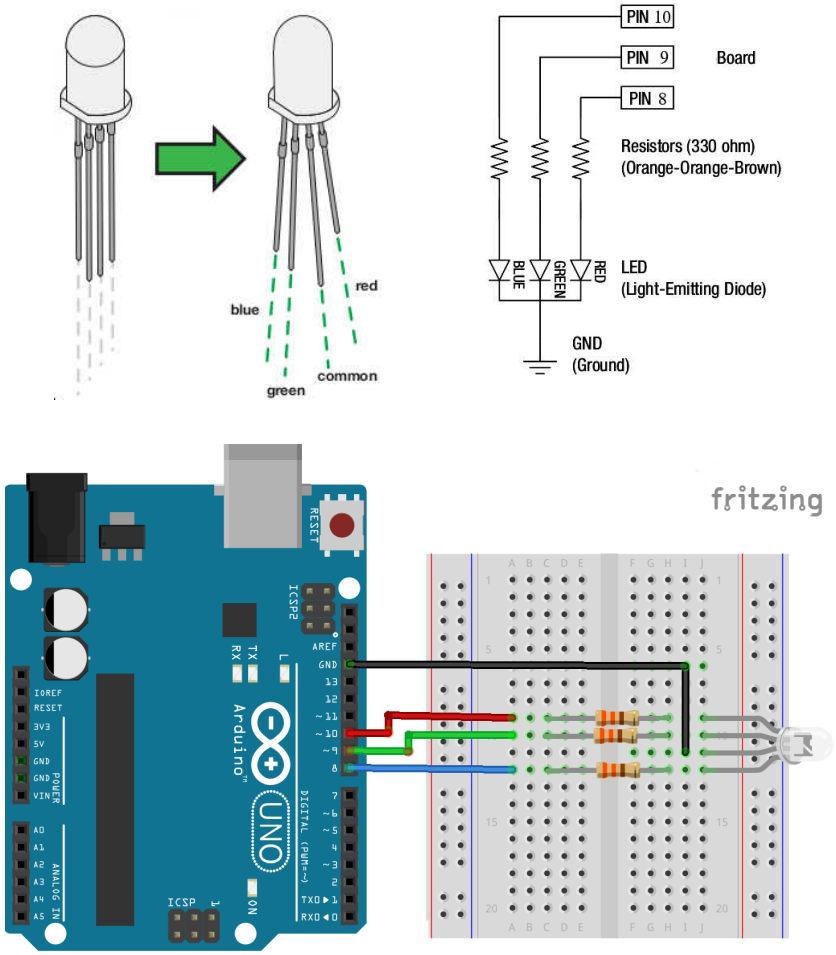
Nous allons commencer avec deux exemples très simples pour mettre en évidence tout le potentiel de cette carte dans les expériences de physique ou de chimie au lycée.

- Le premier exemple nous permettra de commander en allumage une diode RGB grâce aux **entrées/sorties numériques**
- Le deuxième concerne l'utilisation d'une **entrée analogique** propre à la réception d'informations lors de la mise en fonctionnement d'un capteur.

4.1 Synthèse additive des couleurs avec un actionneur de type diode électroluminescente RGB

Dans un premier temps, l'objectif est d'utiliser une DEL capable de produire trois couleurs différentes et de réaliser une synthèse additive avec les couleurs rouge, vert et bleu.

4.1.1 Les schémas électriques (Documentation Arduino)



4.1.2 La réalisation

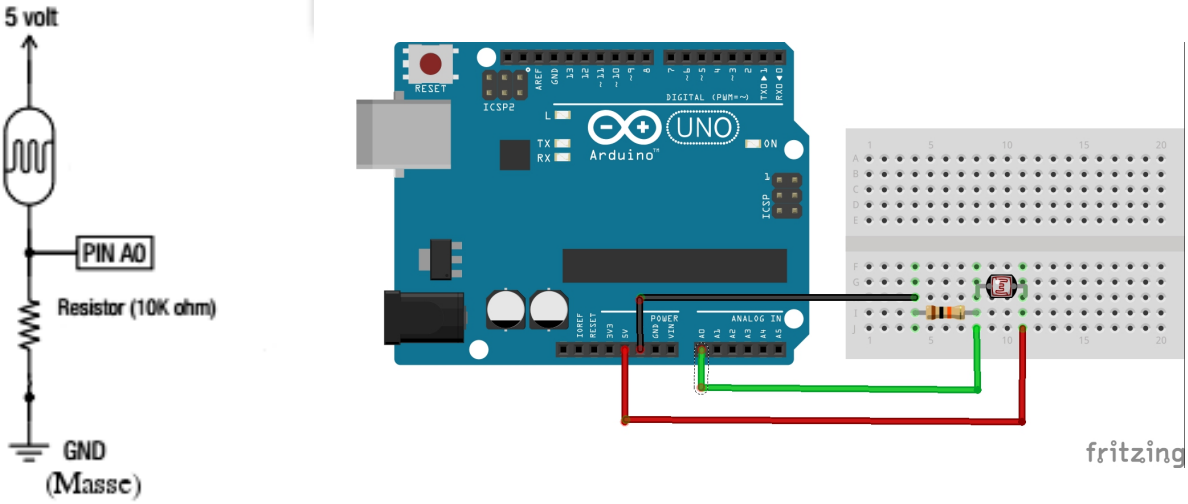
13. À partir du programme test (cf 2.5), écrire un programme Arduino permettant de faire clignoter la LED avec une fréquence de 1 Hz et avec une alternance des couleurs Rouge, Vert, Bleu.
14. Modifier votre programme pour obtenir le clignotement de la LED mais avec l'alternance de couleurs Jaune, Cyan, Magenta et Blanc.
15. Combien de couleurs peut-on obtenir en utilisant la fonction `digitalWrite`?
16. Il est possible d'augmenter le nombre de couleurs grâce à la fonction : `analogWrite`. Lire l'API Arduino pour comprendre l'utilisation de cette fonction.

17. Écrire un programme Arduino permettant d'obtenir quatre niveaux d'intensité différents sur un même canal (0, 75, 125, 250) avec un changement toutes les secondes. Attention il faudra peut-être modifier la position de votre DEL suivant vos attentes. En effet seules les broches précédées d'un ~ sont utilisables avec la fonction : `analogWrite`
18. Combien de couleurs peut-on obtenir avec la fonction : `analogWrite`?

4.2 Mesure de fréquence avec capteur analogique de type photorésistor ou photodiode

4.2.1 Le montage électrique

D'après la doc Arduino



4.2.2 Mesurer la fréquence d'un stroboscope (application smartphone)

Deux possibilités s'offrent à nous. Soit on utilise un stroboscope classique que l'on trouve normalement au laboratoire de physique ou bien il peut être remplacé par une application smartphone.

- Télécharger n'importe quelle application de stroboscope sur votre smartphone, pour cette expérience j'ai utilisé Stroboscopique
- ou télécharger l'application Physics Toolbox Suite, puis cliquer sur stroboscope dans le menu de gauche.
- Régler la fréquence sur 1 Hz
- Placer le flash de votre téléphone au dessus de la photorésistance ou de la photodiode

Le code Arduino associé à cette expérience est extrêmement simple, il nous suffit juste de lire les valeurs envoyées par le capteur (photorésistance ou photodiode) sur une des entrées analogiques de la carte Arduino. Rappelez-vous, celle-ci propose 6 entrées analogiques (de A0 à A5) connectées à un convertisseur analogique-numérique 10 bits (2^{10} valeurs possibles de 0 à $2^{10}-1$). Ce qui permet de transformer la tension d'entrée comprise entre 0 et 5V en une valeur numérique entière comprise entre 0 et 1023. La résolution (écart entre 2 mesures) est de : 5 volts / 2^{10} intervalles, soit 0.0049 volts (4.9 mV).

19. Compléter puis téléverser le code Arduino.
20. Comment faudrait-il modifier le code pour que le nom de variable `valeur` référence bien une tension (attention au type de la variable).

Code Arduino

```
// Variables à déclarer

void setup(){
  Serial.begin(19200);
}

void loop(){
  // À compléter            // valeur numérique lue sur la broche A0
  Serial.print(valeur);   // Envoi la mesure au PC par la liaison série (port USB)
  Serial.print("\t");       // Ajout d'un espace
  Serial.println(millis()); // Envoi de la valeur temps puis retour à la ligne
                            // Une éventuelle temporisation
}
```

21. À l'aide du moniteur série, observer les résultats obtenus.
22. À l'aide d'un tableur, comment tracer le graphique correspondant à $u = f(t)$?

Nous allons maintenant utiliser Python pour automatiser la gestion des données envoyées par le capteur. Pour cela il faut commencer par ouvrir un nouveau notebook que l'on pourra renommer : stroboscope

Dans la première cellule de code recopier les importations des packages nécessaires à la gestion de cet exemple.

Code Python

Les déclarations de packages.

```
1 import serial                    # gestion du port série
2 import time                      # module de gestion du temps
3 import matplotlib.pyplot as plt  # pour faire de beaux graphiques
4
5 # permet d'afficher les graphiques directement dans le notebook
6 %matplotlib inline
```

Dans une deuxième cellule nous allons nous concentrer sur l'écriture du code principal permettant de récupérer et d'organiser les données. Attention la connexion Python avec le port série demande à être adaptée en fonction de votre système d'explotation (3.4.2).

```
1  # connexion Linux au port série
2  serial_port = serial.Serial( port = "/dev/ttyACM1", baudrate =19200)
3  serial_port.setDTR(False)
4  time.sleep(0.1)
5  serial_port.setDTR(True)
6  serial_port.flushInput()
7
8  # les mesures
9  mesure = []
10 temps = []
11 serial_port.flushInput()
12 for i in range(1000):
13     val = serial_port.readline().split()
14     try:
15         t = float(val[1])
16         m = float(val[0])
17         temps.append(t)
18         mesure.append(m)
19     except:
20         pass
21
22 # fermeture du port série
23 serial_port.close()
```

- lignes 9 et 10 : déclaration de deux listes qui recevront les mesures de l'expérience (comme dans les colonnes d'un tableur).
- ligne 11 : On vide la mémoire tampon de la liaison série. Cela permet d'effacer d'éventuelles données de l'acquisition précendente restées en mémoire.
- ligne 12 : On démarre une boucle qui permet de récupérer 1 000 valeurs. Toutes les instructions associées à la boucle sont indentées.
- ligne 13 : permet de lire le flux de données envoyé sur le port série par Arduino. Rappelez-vous le programme Arduino envoie deux valeurs sur le port série, le temps et la mesure du capteur. Il faut donc séparer ces deux valeurs. Pour cela nous utilisons la fonction `split()`. Elle sépare les valeurs grâce à l'espace que nous avons laissé et les rangent dans une liste Python. Une liste Python peut-être vue comme un tableau dont chaque case porte un numéro.
  - La première case a le numéro 0.
  - Pour ajouter une valeur dans la liste on utilise la fonction `append()`

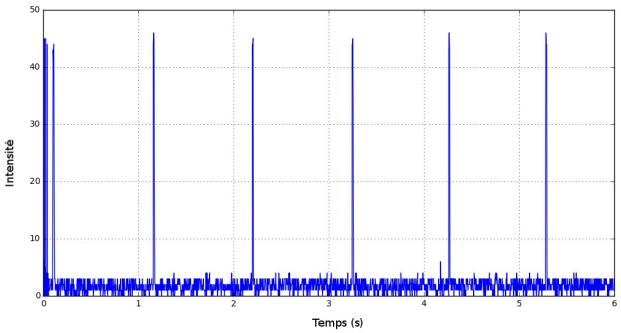
  - Pour lire la valeur contenu dans la première case de la liste `val` on écrit : `val[0]`, pour lire la valeur contenue dans la n ième case on écrit : `val[n]`, pour lire les valeurs comprises entre les cases n et m incluses on écrit : `val[n : m+1]`
- ligne 14 : **try** permet de gérer une erreur d'exécution dans un programme sans pour autant que le programme s'arrête brutalement.
  Le mécanisme de gestion des exceptions s'effectue en deux phases :
  - La levée d'exception avec la détection d'erreurs : le problème se trouve lignes 15 et 16 lors de la conversion.
  - Le traitement approprié : ligne 20 nous décidons de ne rien faire avec le mot clé **pass**
- ligne 15 à 18 : Nous essayons de convertir les données reçues en valeurs décimales et nous les ajoutons aux listes `temps` et `mesure` . N'oublions pas que les données envoyées par Arduino sont au format texte. Il est donc nécessaire de les convertir en valeur numérique. La conversion réalisée est de type *float* soit des valeurs décimales.
- ligne 19 et 20 : Si cela ne marche pas, on passe et on lit une nouvelle ligne envoyée par Arduino

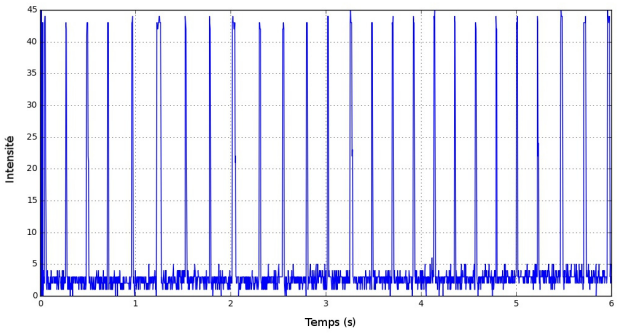
Normalement plus de mystère vous pouvez maintenant recopier le code correspondant à l'acquisition des mesures. Le gros avantage c'est que l'on écrit le code une fois pour toute. Il peut même être déja disponible dans un notebook que vous donnez à vos élèves. Mais rien n'empêche de le modifier pour satisfaire une demande particulière.

L'affichage sous la forme d'un graphique

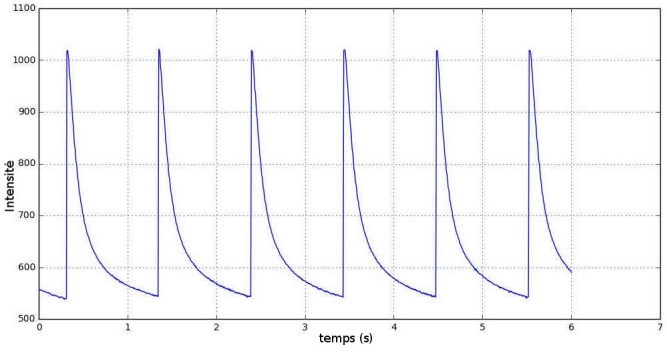
```
1 # On évite les effets de bord en éliminant
2 #les valeurs de début et de fin de transmission
3 plt.plot(temps[100:900], mesure[100:900])
4 plt.xlabel("Temps (s)")
5 plt.ylabel("Intensité")
6 plt.grid()
7 plt.show()
```

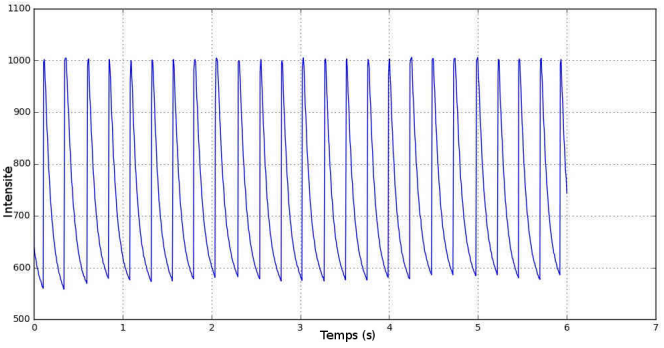
Photodiode (f = 1Hz et 4Hz)





Photorésistance (f = 1Hz et 5Hz)





On voit qu'après chaque flash (supposé suffisamment court), le photorécepteur reste conducteur pendant une durée qui va dépendre du type de photorécepteur

- pour la photodiode temps de réponse très court de quelques microsecondes. Cela illustre la bonne réponse en fréquence de la photodiode.
- pour la photoresistance un temps de réponse relativement court mais elle reste conductrice durant plusieurs dixièmes de secondes

On peut améliorer la lecture du flux de données afin d'assouplir l'utilisation du code Python. Pour cela nous allons écrire deux fonctions Python dont nous détaillerons l'utilisation.

```python
def acquisition(n, serial_port):
    '''
    Cette fonction permet de faire l'acqusition des données
    en fonction du temps reçues par le port USB.
    Elle renvoie deux listes :  temps et mesures (du capteur)

    n           <int>    : nombre total de valeurs à lire
    serial_port <serial> : le port série ouvert à la communication
    '''
    i = 0
    temps, mesures = [], []
    while i < n:
        val = serial_port.readline().split()
        try:
            t = float(val[1])
            m = float(val[0])
            temps.append(t)
            mesure.append(m)
            i = i + 1
        except:
            pass
    return temps, mesures
```

23. Comment le code de la fonction `acquisition` a t-il été modifié par rapport au code précédent et pourquoi?

Pour lancer une acquisition avec 1 000 points :

```python
# connexion Linux au port série
serial_port = serial.Serial( port = "/dev/ttyACM1", baudrate =19200)
serial_port.setDTR(False)
time.sleep(0.1)           # attention le module time est nécessaire
serial_port.setDTR(True)
serial_port.flushInput()

temps, mesure = acquisition(1000, serial_port)

# fermeture du port série
serial_port.close()
```

4.2.3 Fixer la durée d'acquisition

Dans l'exemple précédent l'acquisition dépend d'un nombre de points. Mais il est souvent plus utile de pouvoir contrôler le temps d'acquisition. Le code Arduino ne change pas et le code Python ne va subir qu'une toute petite modification au niveau de la boucle. Au lieu de compter un nombre de points nous allons définir un temps d'acquisition. Rappelons que le code Arduino transmet à chaque itération de la fonction `loop` une ligne contenant une **valeur** et une **date** d'acquisition. Pour contrôler le temps d'acquisition il suffit donc de surveiller la différence entre la date en cours d'acquisition et la date du début d'acquisition. Comme les dates d'acquisition sont dans une liste `temps`, nous allons surveiller `temps[-1] - temps[0]` avec :

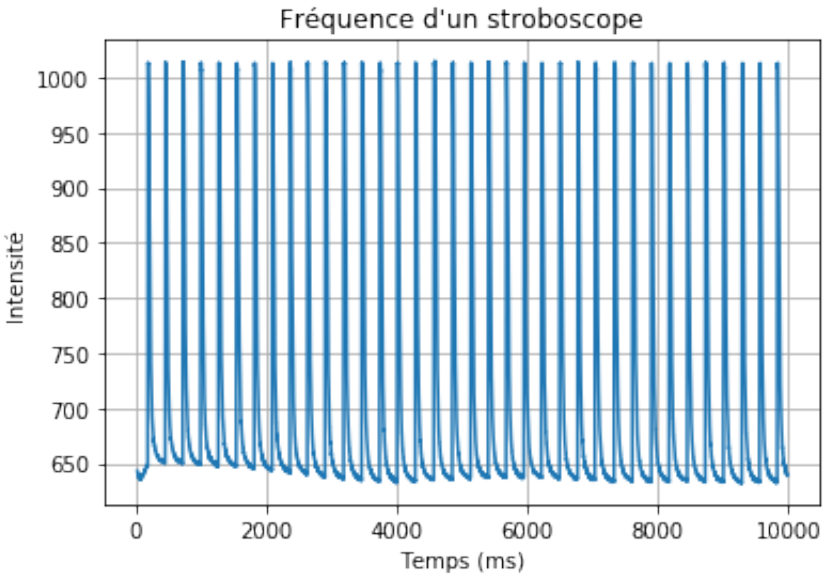
- `temps[-1]` le dernier élément de la liste `temps`
- `temps[0]` le premier élément de la liste

```
# ouverture du port série
serial_port = serial.Serial( port = "/dev/ttyACM0", baudrate =19200)
serial_port.setDTR(False)
time.sleep(0.1)
serial_port.setDTR(True)
serial_port.flushInput()

# les mesures
mesure = []
temps = []
duree = 10000
end = False

while end == False or temps[-1] - temps[0] <= duree:
    val = serial_port.readline().split()
    try:
        t = float(val[1])
        m = float(val[0])
        temps.append(t)
        mesure.append(m)
        end = True
    except:
        pass
# fermeture du port série
serial_port.close()
```

24. Écrire une fonction `acquisition_temps(duree, serial_port)` qui prend en paramètres la durée d'acquisition et la connexion au port série. Cette fonction renvoie dans l'ordre la liste des dates et mesures de l'acquisition.

L'affichage sous la forme d'un graphique

```
# attention les deux listes doivent contenir le même nombre de valeurs.
plt.plot(temps, mesure)

plt.title("Fréquence d'un stroboscope")
plt.ylabel('Intensité')
plt.xlabel('Temps (ms)')
plt.grid()
plt.show()
```



4.3 Utilisation d'un bouton poussoir pour déclencher l'acquisition

L'objectif est **d'ajouter à l'expérience du stroboscope, un bouton poussoir** pour déclencher l'acquisition coté Arduino afin que Python puisse enregistrer les données transférées. Dans cet exemple, très fortement inspiré d'une activité de Jean-Luc Charles [5], nous parlerons **d'automate**.

Concept d'automate

Un automate fini est un modèle mathématique des systèmes ayant un nombre fini d'états et que des actions (externes ou internes) peuvent faire passer d'un état à un autre.

Rien de mieux qu'un exemple pour comprendre :

- à l'état initial, l'automate est à l'état WAIT : l'acquisition est en attente,
- l'appui sur le bouton poussoir fait passer l'automate dans l'état START : l'acquisition démarre,
- un nouvel appui sur le bouton poussoir fait passer l'automate de l'état START à l'état STOP : l'acquisition est suspendue,
- les appuis successifs font passer successivement de l'état START à l'état STOP, et de l'état STOP à l'état START.

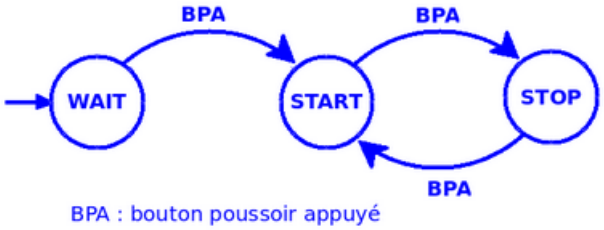
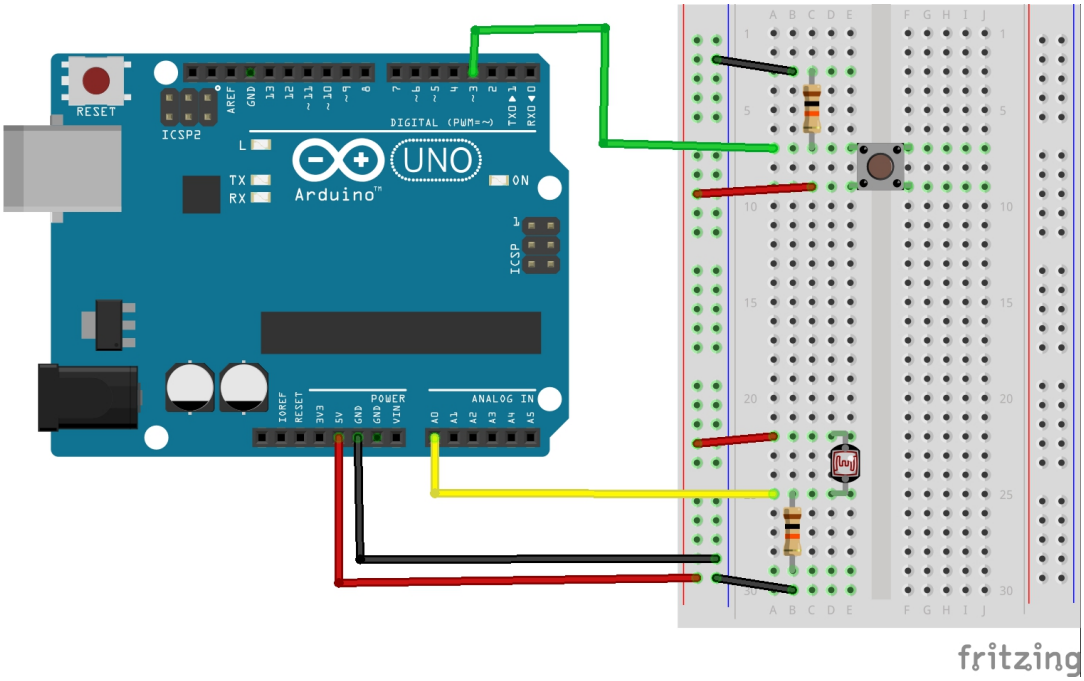


Image extraite d'une activité de Jean-Luc Charles (note : 5)

4.3.1 Le montage électrique



La broche numérique 3 de la carte Arduino est utilisée comme une entrée numérique qui reste à LOW tant que le bouton n'est pas enfoncé. Le bouton se comporte comme un intérrupteur qui ne laisse pas passer le courant tant qu'il est en position haute. Dans cet exemple la broche 3 est en mode INPUT : `pinMode(3, INPUT)`, pour indiquer que la broche est en mode lecture. **Elle ne va donc pas piloter du courant, mais être à l'écoute du courant qui lui arrive.**

5. Jean-Luc Charles, enseignant chercheur à l'ENSAM Talence

4.3.2 Le code Arduino

Coté Arduino ça donne quoi? Commençons par les variables globales et la fonction `setup`

```
1  // Etat en cours de l'automate
2  int etat;
3  // Etat à mémoriser
4  int oldEtat;
5
6  //Les états possibles de l'automate
7  const int WAIT = 2;
8  const int START = 1;
9  const int STOP  = 0;
10
11 // Les broches utilisées
12 //capteur
13 const int broche = A0;
14 //bouton poussoir
15 const int BP = 3;
16
17 void setup() {
18   //initialisation des variables
19   oldEtat = LOW;
20   etat = WAIT;
21   //config E/S
22   pinMode(BP, INPUT);
23   //liaison série
24   Serial.begin(19200);
25 }
```

Comme convenu dans l'introduction nous avons défini les différents états de notre automate et initialisé une variable `oldEtatBP` qui nous permettra de garder en mémoire l'état du bouton avant un nouvel appui. On remarquera également que l'état de notre automate est `WAIT`, nous attendons le démarrage de l'acquisition.

```
1  void loop() {
2    int etatBP = digitalRead(BP);            // Lecture du bouton
3    if(oldEtat == LOW && etatBP == HIGH){    //gestion des états
4      if (etat == WAIT)
5      {
6        etat = START;
7      }
8      else if (etat == STOP)
9      {
10       etat = START;
11     }
12     else if (etat == START)
13     {
14       etat = STOP;
15     }
16   }
17   //Traitement des états
18   if(etat == START){
19     int valeur = analogRead(broche);
20     Serial.print(valeur);
21     Serial.print("\t");
22     Serial.println(millis());
23   }
24   oldEtat = etatBP;
25   delay(10);
26 }
```

Il n'y a plus qu'à tester le programme :

- Téléverser le programme dans la mémoire de la carte Arduino.
- Ouvrir le moniteur série.
- Lancer l'acquisition en appuyant une première fois sur le bouton
- Stopper l'acquisition en appuyant une deuxième fois sur le bouton.
- On peut recommencer autant de fois que l'on veut...

25. Modifier le programme pour que lorsque l'acquisition s'arrête, c'est à dire que l'on appuie sur le bouton pour la deuxième fois, la chaîne `-1\t -1` s'affiche dans le moniteur série. Attention dans le moniteur série le `\t` sera remplacé par une tabulation.

4.3.3 Le code Python

Attention le code Arduino ci-dessous fonctionnera correctement uniquement si vous avez répondu à la question précédente, si cela pose un problème consulter la solution dans les annexes, compléter le code Arduino et poursuiver.

```
# les modules à importer
import serial
import matplotlib.pyplot as plt
```

```
# ouverture du port série et synchronisation des données entre arduino et Python.
serial_port = serial.Serial( port = "/dev/ttyACM0", baudrate =19200, timeout = None)
serial_port.flushInput()

# les mesures
mesure = []
temps = []
end = False

while end == False:
    val = serial_port.readline().split()
    if val[0] == b'-1':                    # Bouton poussoir à l'état STOP
        end = True                         # Terminaison de la boucle
    else:
        try:
            t = float(val[1])
            m = float(val[0])
            temps.append(t)
            mesure.append(m)
        except:
            pass

# fermeture du port série
serial_port.close()
```

Pour tester l'ensemble, assurez-vous que vous avez bien effectué les étapes de la section précédente coté Arduino :

- Valider les cellules du Notebook.
- Normalement sur la gauche de la deuxième cellule vous observez une petite étoile : **In [*]**
- Pas de panique avec le bouton on a tout notre temps.
- Positionner votre stroboscope au dessus de la photorésistance
- Lancer l'acquisition des valeurs en appuyant une première fois sur le bouton.
- Terminer l'acquisition en appuyant une deuxième fois sur le bouton, si tout c'est bien passé l'étoile de votre **In [*]** disparait pour laisser place à un nombre.
- Afficher vos résultats dans un graphique.

À chaque fois que l'on termine une acquisition il faut revalider la cellule du notebook contenant le code ci-dessus pour mettre en attente le code Python d'une nouvelle acquisition. L'instruction `serial_port.close()` réinitialise le code Arduino et met donc l'automate coté Arduino dans l'état `WAIT`. Il n'y a plus qu'à appuyer sur le bouton...

4.4 Temps de réponse d'une thermistance de type CTN

4.4.1 Présentation de la thermistance CTN

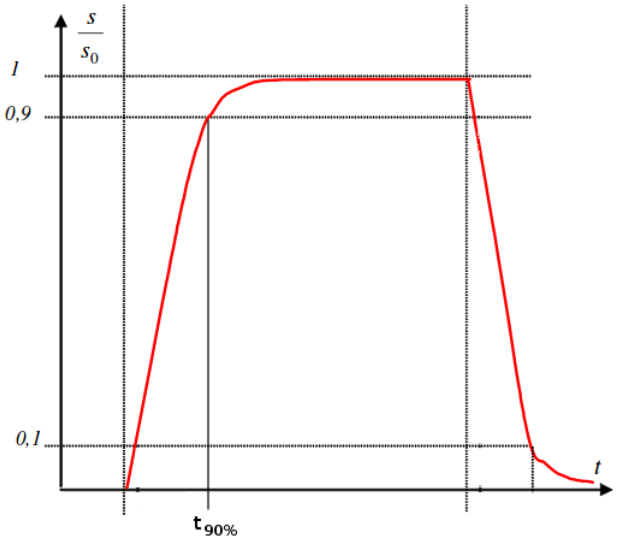
La dépendance en température d'une résistance CTN n'est pas linéaire, elle peut être bien approximée en fonction de la résistance R_{Th} de la thermistance à l'aide de la relation suivante :

$$\theta = \frac{1}{\frac{\ln\left(\frac{R_{Th}}{R_0}\right)}{\beta} + \frac{1}{T_0}} - 273.15$$

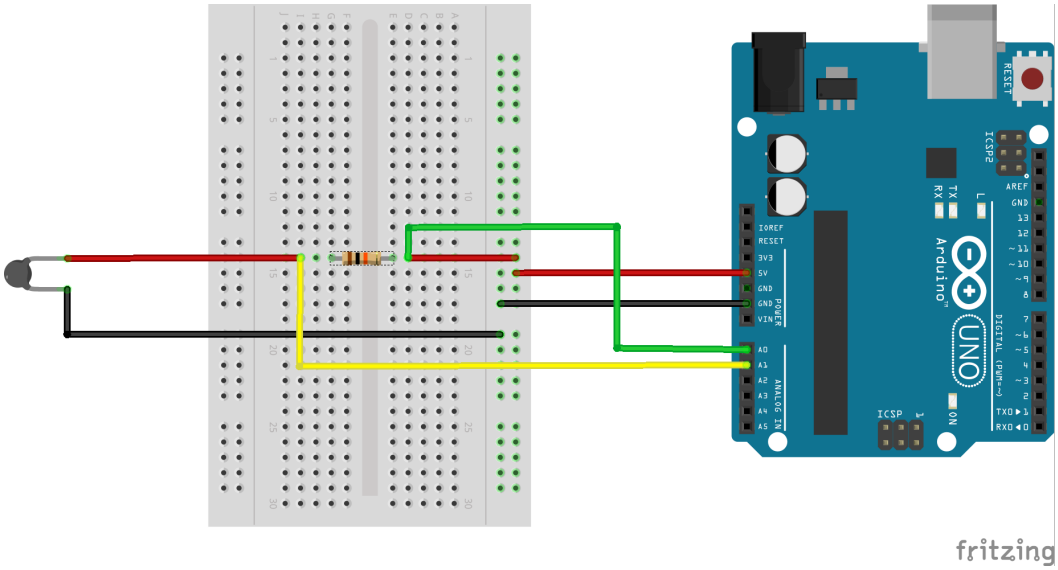
β est une constante de température (kelvin), R_0 est une résistance de référence (ohms) et T_0 est la température à laquelle s'applique la résistance de référence (kevlin). Ces constantes sont caracteristiques de la thermistance utilisée.

Le temps de réponse d'un capteur en température n'est pas nul car le capteur ne s'adapte jamais instantanément à une variation de température du milieu ambiant (air, eau, huile moteur, ...). Si la température du milieu ambiant passe d'une valeur θ initiale à une valeur θ finale supérieure à la température initiale θ initiale , le temps de réponse $t_{90\%}$ est la durée nécessaire pour que la température mesurée par le capteur passe de la valeur θ initiale à la valeur :

$$\theta = \theta_{\text{initiale}} + 0,9 \times (\theta_{\text{finale}} - \theta_{\text{initiale}})$$



4.4.2 Le montage électrique



Il est possible d'acheter des thermistances CTN (10K) étanches pour un prix très raisonnable (< 1 € ref : NTC Thermistance précision capteur de température 10 K 1% 3950 Sonde Étanche 1 m)

- $\beta = 3380\,\text{K} \pm 1\%$
- $R_0 = 10\,\text{k}\Omega \pm 1\%$
- plage de mesure :-20 à 105 ^{o}C avec $T_0 \simeq 298\,\text{K}$

4.4.3 Les codes du montage

Coté Arduino

Le code à compléter ci-dessous peut-être l'occasion de discuter de la conversion d'une valeur numérique codée sur 10 bits (A0 et A1) en valeur analogique. Pour compléter ce code nous pourrons utiliser la fonction `map`.

La fonction `map` avec Arduino

Ré-étalonne un nombre appartenant à un intervalle de valeurs vers un autre intervalle de valeurs. Dans notre cas la valeur numérique $\in [0, 1023]$ en valeur analogique $\in [0\,\text{V}, 5\,\text{V}]$

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

- valeur : le nombre à ré-étalonner
- limite_basse_source : la valeur de la limite inférieure de la fourchette de départ
- limite_haute_source : la valeur de la limite supérieure de la fourchette de départ
- limite_basse_destination : la valeur de la limite inférieure de la fourchette de destination
- limite_haute_destination : la valeur de la limite supérieure de la fourchette de destination

Il est possible d'obtenir des informations supplémentaires et des exemples liés à cette fonction avec l'API d'Arduino.

```
1  //Fonction setup(), appelée au démarrage de la carte Arduino
2  void setup()
3  {
4    Serial.begin(9600); // initialisation de la communication série à 9600 bps
5  }
6  // Fonction loop(), appelée en boucle si la carte Arduino est alimentée
7  void loop()
8  {
9    // Declaration des variables
10   unsigned long temps=millis();
11   double U, Uther, tensionU, tensionUther;
12   // lecture des tensions U et Uther sur A0 et A1 et initialisation
13   // du compteur temporel
14   temps = millis()/1000;
15   U = analogRead(0) ;
16   // conversion de la tension lue sur A0 en valeur analogique
17   tensionU =
18   // conversion de la tension lue sur A0 de mV en V
19   tensionU =
20
21   Uther = analogRead(1) ;
22   // conversion de la tension lue sur A1 en valeur analogique
23   tensionUther =
24   // conversion de la tension lue sur A1 de mV en V
25   tensionUther =
26
27   // Envoi les mesures sur le port série
28   Serial.print(tensionU);
29   Serial.print("\t");
30   Serial.print(tensionUther);
31   Serial.print("\t");
32   Serial.println(temps);
33
34   // intervalle de temps d'une seconde (1000 ms) entre
35   // deux executions de la  boucle donc entre deux mesures
36   delay(1000);
37 }
```

Coté Python

```
1 # Cellule 1 : les modules à importer
2 import serial
3 import time
4 import numpy as np
5 import matplotlib.pyplot as plt
6 %matplotlib auto
```

Dans la cellule des modules à importer rien de nouveau à part la ligne 6. Dans les exemples précédents, nous avions l'habitude d'écrire `%matplotlib inline`. Nous avons remplacé `inline` par `auto` afin de pouvoir afficher une fenêtre extérieure au Notebook. Cette fenêtre offre quelques fonctionnalités de base étendues comme la possibilité de suivre la courbe avec un réticule.

La cellule suivante donne la définition d'une fonction permettant d'effectuer la synchronisation temporelle pour le transfert des valeurs entre Arduino et Python. Les instructions liées à cette fonction ont déjà été décrites dans la partie *Communication Arduino - Python via le port série*

```
1  # Cellule 2
2  def synchro_arduino(port_name, vitesse):
3      """
4      Cette fonction initialise et renvoie une référence sur la connexion
5      avec la carte arduino à travers le port série (USB).
6      Elle permet également de synchroniser le lancement de l'acquisition
7      coté Arduino avec la récupération des données coté Python.
8      """
9      serial_port = serial.Serial( port = port_name, baudrate =vitesse)
10     serial_port.setDTR(False)
11     time.sleep(0.1)
12     serial_port.setDTR(True)
13     serial_port.flushInput()
14     return serial_port
```

À vous de compléter la fonction `modelisation_CTN` afin de calculer la température correspondante aux mesures reçues par Python.

```
1  # Cellule 3
2  def modelisation_CTN(mesures):
3      """
4      Cette fonction réalise le traitement des données, associées au capteur
5      thermistance, reçues de la carte Arduino.
6      Elle renvoie :
7          tensionU     -> float : la tension délivrée par la carte Arduino
8          tensionUther -> float : la tension aux bornes du capteur
9          Rther        -> float : la valeur de la résistance de la thermistance
10         temps        -> float : la date de la mesure
11         temperature  -> float : la valeur de la temperature
12     Elle prend en argument la liste des mesures effectuées par Arduino
13         tensionU     -> float
14         tensionUther -> float
15         temps        -> float
16     """
17     Rzero = 10000  # en ohms
18     beta  = 3380   # en Kelvins
19     Tzero = 298    # en Kelvins
20
21     tensionU, tensionUther, temps = mesures
22
23     Rther       = tensionUther*(1/(tensionU-tensionUther)*Rzero)
24     temperature = # À compléter
25
26     return tensionU, tensionUther, Rther, temps, temperature
```

La dernière cellule concerne essentiellement l'affectation des valeurs à afficher à la bonne structure de données, dans notre cas des listes Python. Cette cellule est pratiquement identique à celle des exercices précédents sans le bouton poussoir. Libre au lecteur de l'adapter s'il en ressent le besoin. J'ai juste ajouté le formatage des données pour une bonne lecture dans la sortie du Notebook. J'ai essayé de faire en sorte que cela ressemble à un tableau. On peut faire bien mieux en utilisant un module plus adapté comme Pandas avec ses DataFrames mais cela sortirait du cadre de cette formation.

```
# Cellule 4
# ouverture du port série
serial_port = synchro_arduino("/dev/ttyACM0", 9600)

# les mesures
temperature = []
temps       = []
duree       = 100     # en seconde (1min 40s)
end         = False

# On s'occupe de l'affichage des résultats
head = "tensionU\t tensionUther\tRther\ttemps\ttemperature\n"
print(head.expandtabs(10))
fmt = "{0:.2f}\t{1:.2f}\t{2:.2f}\t{3:.2f}\t{4:.2f}".expandtabs(16)

# on récupère les données et on modélise
while end == False or temps[-1] - temps[0] < duree:
    data_arduino = serial_port.readline().split()
    try:
        mesures = np.array(data_arduino, dtype='float') # String -> flottant
        mesures = modelisation_CTN(mesures)             # Calcul température
        temps.append(mesures[3])            # remplissage liste des temps
        temperature.append(mesures[4])      # remplissage liste des températures
        print(fmt.format(*mesures))         # formatage de l'affichage
        end = True
    except:
        pass
# fermeture du port série
serial_port.close()
```

4.4.4 L'expérience et ses résultats

De nombreuses expériences sont possibles, pour ma part j'ai déposé quelques glaçons dans le fond d'un premier récipient avec un peu d'eau puis j'ai rempli un deuxième récipient avec un peu d'eau à la température de la pièce.

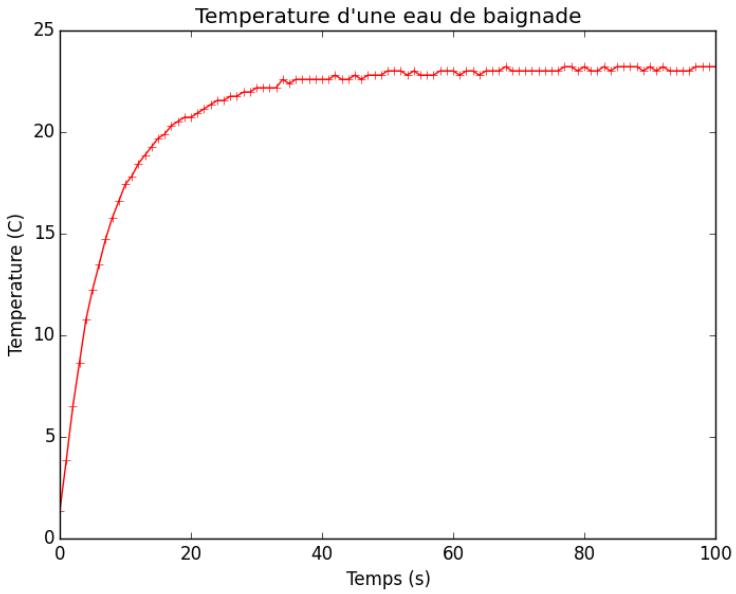
- Téléverser le code Arduino dans la mémoire de la carte
- Valider les cellules 1, 2 et 3
- Plonger la thermistance environ 30 secondes dans le récipient avec les glaçons
- Plonger la thermistance dans l'eau du deuxième récipient puis valider la cellule 4

Voici les résultats obtenus dans le Notebook

```
# fermeture du port série
serial_port.close()
```

tensionU	tensionUther	Rther	temps	temperature
5.00	3.70	28461.54	0.00	1.35
5.00	3.59	25460.99	1.00	3.85
5.00	3.47	22679.74	2.00	6.50
5.00	3.37	20674.85	3.00	8.65
5.00	3.27	18901.73	4.00	10.77
5.00	3.20	17777.78	5.00	12.24
5.00	3.14	16881.72	6.00	13.49
5.00	3.08	16041.67	7.00	14.74
5.00	3.03	15380.71	8.00	15.77
5.00	2.99	14875.62	9.00	16.60
5.00	2.95	14390.24	10.00	17.42
5.00	2.93	14154.59	11.00	17.84
5.00	2.90	13809.52	12.00	18.46

Le graphique : Temperature = f(Temps)



Exploitation

Le temps de réponse $t_{90\%}$ peut ainsi être calculé facilement avec Python

```
1 Tinitial = temperature[0]                    # première valeur dans la liste
2 Tfinal = np.mean(temperature[60:-1])         # une moyenne sur les dernières valeurs
3 T = Tinitial + 0.9 * (Tfinal - Tinitial)     # température à t90%
4 print('Tinitial = {0:.2f} C ; Tfinal = {1:.2f} C'.format(Tinitial, Tfinal))
5 print('T(t90%) = {0:.2f} C'.format(T))
```

```
Tinitial = 1.35°C ; Tfinal = 23.08°C
T(t90%) = 20.90°C
```

On peut ainsi facilement tracer la droite $Temperature = T$ avec l'instruction
`plt.axhline(y=T,color='b')`
Le réticule permet de lire la valeur de $t \simeq 20.7s$