Utilisation des microcontrôleurs en Physique-Chimie au lycée.

Initiation à l'EDI Arduino avec une carte UNO

L'EDI Arduino est un logiciel qui regroupe tous les outils nécessaires pour programmer la carte UNO et communiquer avec elle : éditeur de texte pour écrire le code, compilateur, téléverseur, terminal de communication série.
La maîtrise du langage Arduino n'est pas exigible au lycée mais il se comprend bien si les élèves ont des bases en python et si les lignes de code sont bien commentées.

I- Exemple : charge d'un condensateur, acquisition/représentation graphique de $u_C(t)$ et mesure de τ

Objectifs et outils :

Il s'agit d'enregistrer et de représenter au cours du temps la tension aux bornes du condensateur d'un dipôle RC soumis à un échelon de tension, ainsi que de mesurer le temps caractéristique, tout cela à l'aide d'une ***carte de type UNO*** *jouant à la fois le rôle de générateur de tension et d'interface d'acquisition, et de l'****EDI Arduino*** *servant à la programmation puis comme grapheur à l'exécution.*

Lien avec le programme de physique-chimie du lycée :

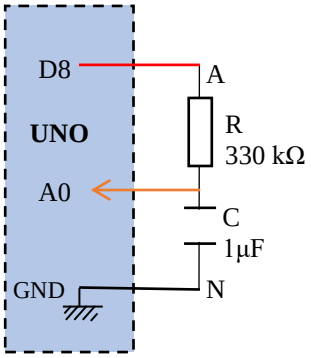
NIVEAU	*Terminale spécialité phys-chim.*
Partie du programme officiel	*Ondes et signaux*
Capacité expérimentale ou numérique	*Étudier la réponse d'un dispositif modélisé par un dipôle RC. Déterminer le temps caractéristique d'un dipôle RC à l'aide d'un microcontrôleur*

Pour un exemple d'exploitation en TP, voir la ressource académique :
Fiche-TP2-Etude-du-dipole-RC-serie-soumis-a-un-echelon-de-tension.pdf

Matériel : *carte type UNO + EDI Arduino, platine d'essais + fils Dupont M/M, résistor et condensateur tels que RC ~ 10^{-1} s .*

Principe du dispositif :

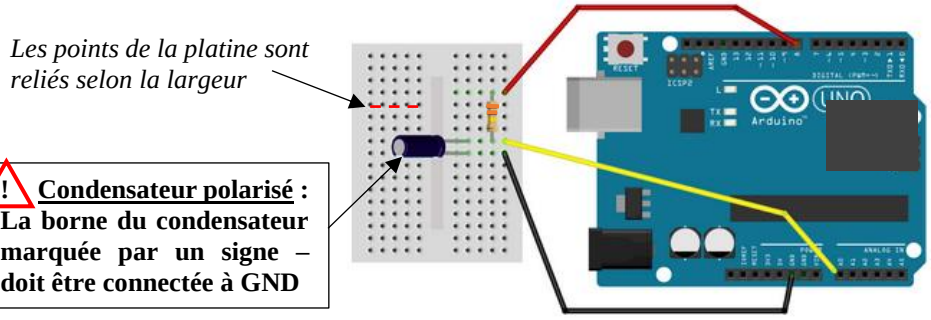
Le condensateur étant déchargé, on souhaite enregistrer sur l'entrée analogique A0 la tension $u_C(t)$ aux bornes du condensateur quand le dipôle série RC est soumis à un échelon de tension de 5V entre la masse et la sortie D8 (cf. schéma ci-contre).



1) Préparation hors connexion du montage sur carte UNO

L'USB de la carte UNO étant **NON connecté**,

(***cf. dessin ci-dessous***)
- réaliser sur platine d'essais le dipôle RC série,
- relier ses bornes aux ports GND (fil noir) et numérique 8 (fil rouge) de la carte UNO,
- connecter l'entrée A0 sur le montage avec un 3e fil coloré

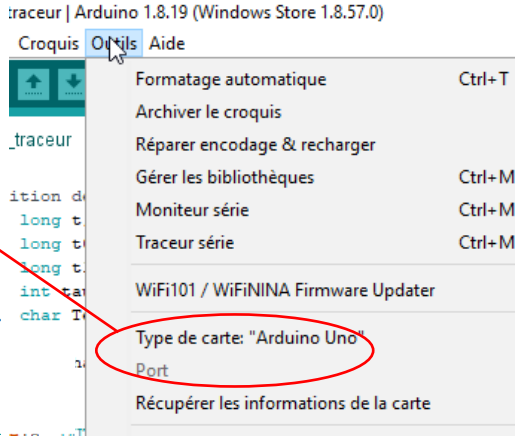


Précaution d'usage : avant de connecter la carte UNO pour la programmer, déconnecter la broche D8. Ne la reconnecter qu'après avoir téléversé le programme.

2) Connection et programmation de la carte

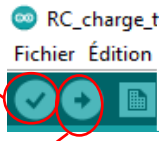
- La session Windows étant ouverte, connecter la carte UNO sur le PC via le câble USB.

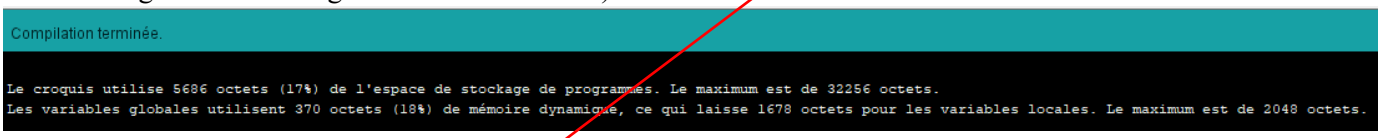
- Télécharger depuis le site académique le fichier ***RC_charge_traceur_.ino*** accompagnant la ***Fiche-TP2*** puis l'ouvrir depuis *Arduino IDE* (qui demandera à créer un dossier du même nom). Si besoin, sélectionner *Afficher les numéros de ligne* dans le menu *Fichier > Préférences*.



- Dans le menu *Outils* : sélectionner le *type de carte > Arduino Uno*, puis sélectionner le *Port > COM* de la carte UNO (en cas de doute, regarder sur quel port COM est branchée la carte UNO dans le menu *périphériques* des *paramètres* de *windows*).

- *Vérifier* le code (la compilation peut durer quelques minutes la 1ère fois) ; le message suivant apparaît si le code est correct (ou un message d'erreur orange dans le cas contraire) :





- *Téléverser* le programme dans la carte UNO.

Copie d'écran du script *RC_charge_traceur_.ino :*

les textes situés après // (ou entre /* et */ pour plusieurs lignes) sont des commentaires destinés à la compréhension du code

```
// Déclaration des contantes
byte Te = 10;           // durée fixe entre deux acquisitions (petite valeur entière en ms)

// Déclaration des variables
unsigned long t;                // temps qui prendra des grandes valeurs entières positives
unsigned long t0;               // origine des temps (début charge)
unsigned long t1 = 0;           // servira à mémoriser l'instant de l'acquisition
unsigned int tau = 0;           // temps caractéristique en ms et en valeur entière positive
float uc;                       // tension uc qui prendra des valeurs décimales
```

Le bloc fonctionnel « setup » n'est exécuté qu'une seule fois

```
void setup() {
  // Initialisation des ports
  pinMode(8, OUTPUT);   // définit la broche numérique 8 comme une sortie de la carte UNO
  Serial.begin(9600);   //ouvre une communication avec le port série de l'ordinateur

  // Dans un premier temps, on s'assure que le condensateur est complètement déchargé
  digitalWrite(8,LOW);            // sortie numérique 8 mise à 0,0V
  Serial.println("Patientez_3s_SVP décharge...");   // affiche un texte via le port série (en légende si traceur OU suivi d'ur
  delay(3000);                    // délai de décharge (valeur à adapter au temps caractéristique théorique RC)

  // Dans un deuxième temps, charge du condensateur
  Serial.println("uc=f(t) charge...");
  uc = float(analogRead(A0))*5/1023; // lit la valeur numérique sur l'entrée A0 et la convertit en volts (0,0-5,0V = 0-1023)
  Serial.println(uc);  // affiche via le port série la valeur de uc (0,0V attendu), suivi d'un passage à la ligne si moniteur
  digitalWrite(8,HIGH); // sortie numérique 8 mise à 5,0V (= 1023)
  t0 = millis();    // définition de l'origine des temps à l'aide de la fonction millis() qui renvoie la
                    //  date en ms de l'horlorge interne d'Arduino prise à partir de sa mise sous tension

  while( analogRead(A0) < 1022 ) { // boucle qui s'exécute tant que uc < 5,0V ;
                                   // éviter 1023 comme valeur de comparaison car parfois cette valeur n'est pas atteinte
    t = (millis() - t0) ;        // mesure du temps écoulé depuis l'origine des temps t0
    if ((t - t1) >= Te) {        // bloc qui ne sera exécuté que si 10ms se sont écoulées depuis t1
      uc = float(analogRead(A0))*5/1023;
      Serial.println(uc);        // affiche via le port série la valeur de uc, suivi d'un passage à la ligne si moniteur OU s
      t1 = t;                    // mémorise l'instant de l'acquisition
      if ((analogRead(A0) >= 0.63*1023) and (tau == 0)) {      // condition de mesure de tau : à 63% de la tension de charge
        tau = (t-5);             // mesure de tau à mi-intervalle avec l'acquisition précédente de uC
      }
    }
  }
  digitalWrite(8,LOW);   // remise à 0,0V de la sortie 8
  Serial.print("uc(V)=f(t(cs)) "); // affiche du texte via le port série
  Serial.print("tau=");    //
  Serial.print(tau);       // affiche la valeur de tau via le port série
  Serial.println("ms+/-5ms"); // affiche du texte via le port série, et passe à la ligne si moniteur
}
```

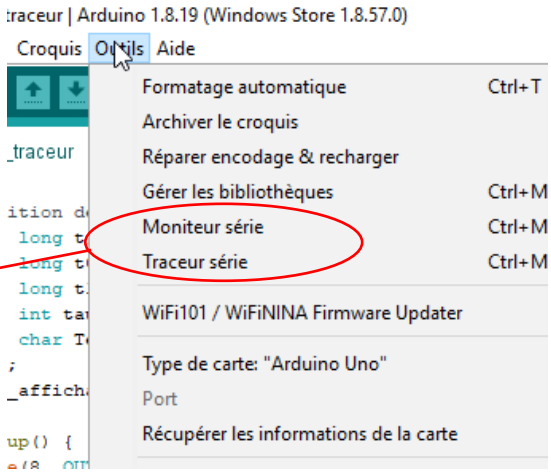
```
53
54 void loop() {
55 }
```

Le bloc fonctionnel « loop » exécute en boucle les instructions qu'il contient (il ne peut être supprimé même s'il est vide)

3) Acquisition et affichage de $u_C(t)$

Exécution du code avec le moniteur série :

Le code n'est pas prévu pour le moniteur série, mais cela permettra de comprendre le fonctionnement de ce dernier.



- Dans le menu *Outils*, ouvrir le ***Moniteur série*** : une fenêtre s'ouvre et le programme téléversé s'exécute :
<u>le moniteur série affiche sous forme de chaînes de caractères (lettres ou chiffres) ce qui est envoyé par le port série</u>, c'est-à-dire ce qui se trouve dans l'argument de l'instruction ***Serial.print().***

Au préalable il a fallu ouvrir une liaison série entre l'ordinateur et le microcontrôleur avec ***Serial.begin(9600)***. La valeur en argument de cette dernière instruction définit la vitesse de communication (*baudrate*) ; ce doit être la même valeur sélectionnée dans la fenêtre du moniteur série ou du traceur série

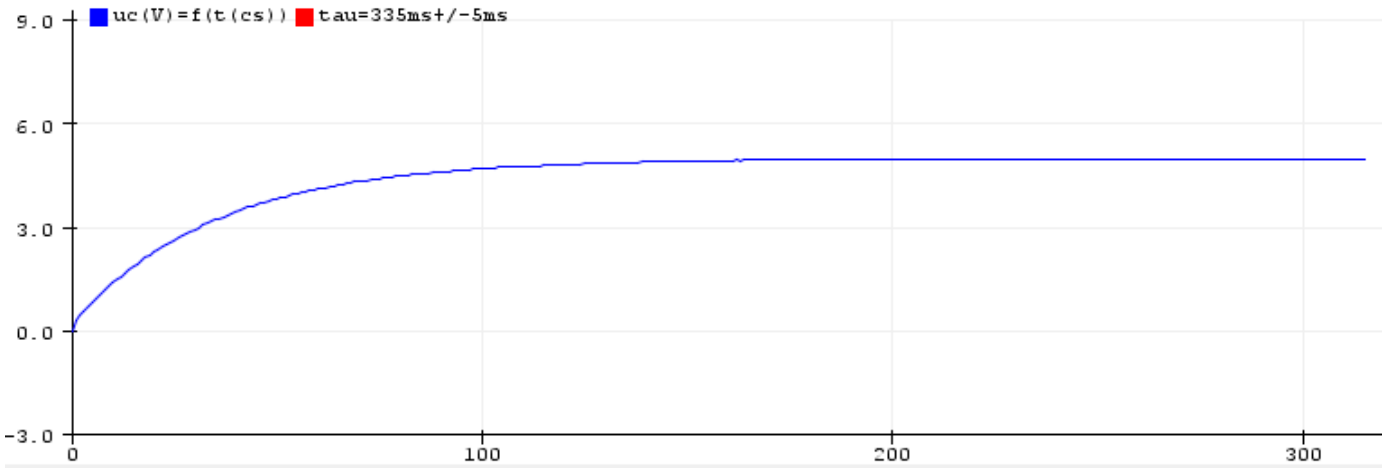
On remarque la variante ***Serial.println()*** qui permet d'aller à la ligne après l'affichage.
Enfin, comme en python, *Serial.print(**tau**)* affiche la valeur de tau (c'est-à-dire les chiffres de l'entier tau), tandis que *Serial.print("**tau**")* affiche le texte tau (c'est-à-dire les lettres du mot tau).

Exécution du code avec le traceur série :

<u>Fermer le Moniteur série</u>, puis, dans le menu *Outils*, ouvrir le ***Traceur série*** : une fenêtre s'ouvre et le programme téléversé s'exécute à nouveau :
<u>l'IDE trace le graphique des points dont les ordonnées sont les valeurs numériques envoyées par le port série avec l'instruction *Serial.println(uc)* (donc ici les valeurs de u_C), en fonction du nombre de valeurs affichées (en abscisse)</u>,
d'où la nécessité dans le code utilisé ici de :
- définir une période d'acquisition (toutes les 10ms ici, ce qui permet d'avoir en abscisse la durée exprimée en cs),
- ne pas afficher les valeurs de t, car sinon elles seraient aussi représentées en ordonnée.

Remarque : sur les anciennes versions de l'EDI Arduino, la légende ne s'affichera pas correctement.

Affichage obtenu à la fin de l'exécution par le traceur série :



4) Que choisir en général pour l'expérimentation : traceur série ou moniteur série ?

- Le traceur série n'a pas d'intérêt pour les utilisations sans graphique (affichages de mesures et de calculs en texte) : dans ces cas le moniteur série s'impose logiquement.

- Pour les acquisitions avec graphique, le traceur série présente l'avantage d'afficher le graphique sans utiliser un autre logiciel, mais on voit vite ses limites : c'est un outil peu pratique (pas de paramétrage, difficulté pour afficher la légende, nécessité d'une acquisition périodique…) qui ne peut servir que pour des acquisitions au cours du temps.

Si on souhaite un graphique d'évolution non temporelle ou si on souhaite ajouter une modélisation, alors il est préférable d'afficher avec le moniteur série les valeurs des abscisses et ordonnées sur 2 colonnes, puis de les exporter au format csv ou txt pour les traiter ensuite avec un tableur ou en python.

II- Utilisation d'un module nécessitant une bibliothèque

Certains capteurs et actionneurs (servomoteurs, écrans, …), en particulier tous les modules grove, nécessitent l'installation d'une bibliothèque (ou paquet) : il s'agit de programmes complémentaires (extensions *.cpp* et *.h*) qui sont appelés dans le code arduino du fichier principal (extension *.ino*) par des instructions "#include … " placées au début du code.

Exemples :

- pour utiliser un écran LCD rgb grove :

```
#include <Wire.h>
#include "rgb_lcd.h"

rgb_lcd lcd;
```

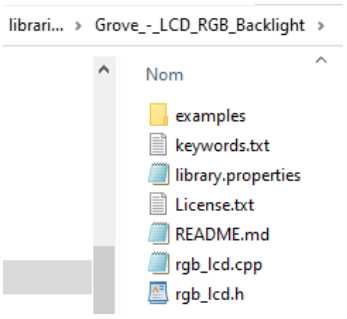
- pour utiliser un servomoteur :

```
#include <Servo.h>

Servo myservo;  // create servo object
```

Les paquets se présentent sous forme de dossiers comprenant les programmes nécessaires mais aussi souvent des informations et des exemples de code réutilisables.

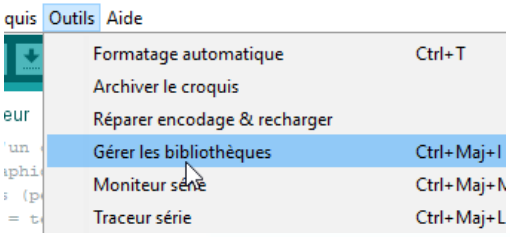
Exemple ci-contre : contenu d'un paquet pour l'utilisation d'un écran LCD rgb grove



Comment installer une bibliothèque ?

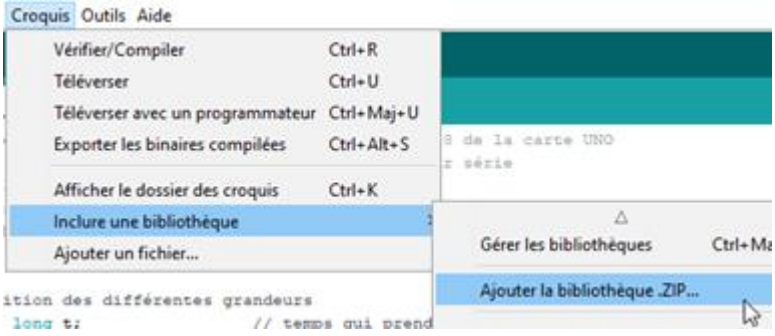
De nombreuses bibliothèques sont déjà automatiquement installées par l'IDE Arduino (en particulier celles communes).

Les bibliothèques installées, celles disponibles et les mises à jour sont affichées et gérées (installation/suppression) en ligne à partir du menu *Outils / Gérer les bibliothèques* (qui lance une fenêtre en recherchant tous les paquets et mises à jour sur internet, ce qui peut être assez long) :



On peut aussi utiliser le menu *Croquis > Inclure une bibliothèque* pour installer une bibliothèque téléchargée :

- les bibliothèques déjà installées y sont aussi visibles
- on peut en ajouter une sous forme de dossier zippé :



https://wiki.seeedstudio.com/How_to_install_Arduino_Library/

III- Où trouver des codes Arduino, des bibliothèques et des schémas de montage ?

L'enseignant de physique-chimie n'a pas besoin de maîtriser le langage Arduino pour coder. Il lui suffit de récupérer des codes existants et de les modifier ou d'en copier/coller des extraits, leur compréhension étant généralement assurée par les commentaires écrits dans les codes par leurs auteurs (éviter les codes non commentés !).

Les sites Arduino dédiés sont très riches en ressources, tout comme bien sûr les sites académiques et Eduscol…

Les informations (codes, bibliothèques, schémas de connexions…) adaptées aux modules utilisés avec la carte UNO peuvent en premier lieu être téléchargées depuis le site du vendeur.