L'Arduino est une carte électronique en Matériel Libre (Open Source) pour la création de prototypage rapide.

I- Préparation du logiciel et de l'interface USB.

Téléchargement du logiciel et configuration de l'ordinateur :

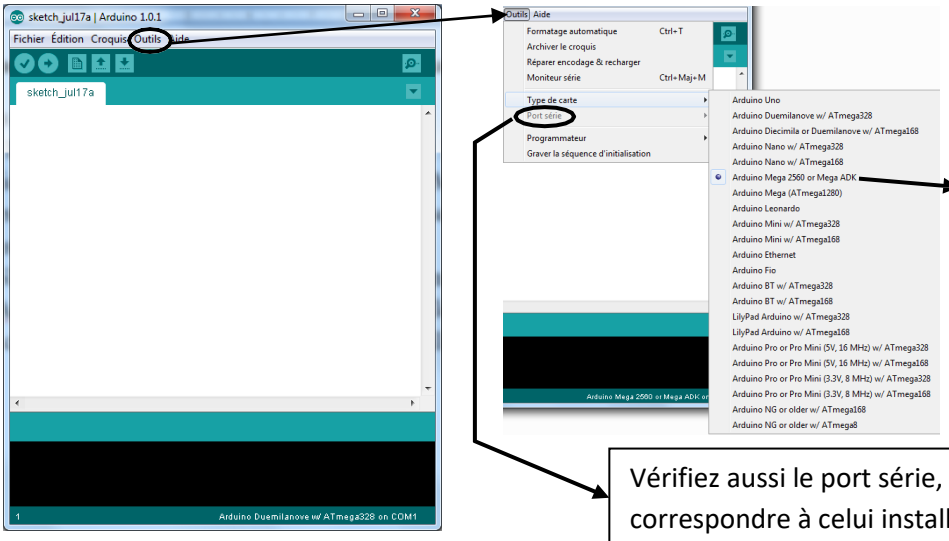
Sur Windows :
-Télécharger la version Windows du logiciel Arduino ici : http://www.arduino.cc/en/Main/ Taille du logiciel : zippé : 80 Mo environ - dézippé : 250 Mo environ
-Le logiciel est prêt à l'emplois. Ce logiciel peut être utiliser en embarqué (sur une clé USB par exemple) sans installation particulière.
-Le pilote USB se trouve dans le répertoire du logiciel dans le répertoire /Drivers/ FTDI USB Drivers
-Brancher l'Arduino et pointer l'installeur Windows vers le pilote
-Et voilà ! la carte est prête à accueillir un programme Utilisateur

Installation du driver USB

Lorsque vous connectez la carte Arduino sur l'ordinateur, celui-ci recherche le driver sans l'obtenir. Allez dans le panneau de configuration et affichez les périphériques.
La carte Arduino doit apparaître mais avec une signalisation vous indiquant que le driver n'est pas installé :

Faites un clique droit et "propriété". Ensuite, faites "Modifier les paramètres". Cliquez sur "Mettre à jour le pilote". La recherche doit se faire "sur l'ordinateur" et dans le répertoire du logiciel : "/arduino-1.0.1/drivers/".
Windows va installez de lui-même le pilote.

Lancement du programme : , trouver dans le répertoire du logiciel le fichier exécutable portant le nom "Arduino". La fenêtre suivante s'affiche :



Dans le menu "Outils", choisissez le type de carte que vous utilisez. Pour nous, ça sera l'"Arduino Mega 2560 or Mega ADK".

Vérifiez aussi le port série, qui doit correspondre à celui installer. Normalement vous n'avez pas le choix (un port supérieur à 2, par exemple "COM 4")

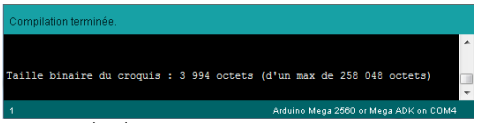
II- Etapes de programmation avec la carte "Arduino"

Etape 1 : ***Création*** du programme

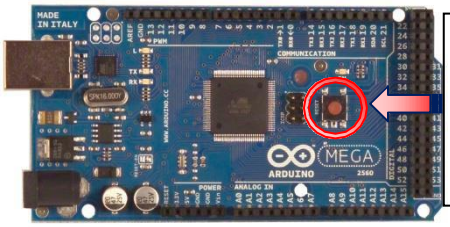
```
/* Ce programme fait clignoter une LED branchée sur la broche 13
* avec une vitesse de clignotement proportionnelle à l'éclairage ambiant
* capté par une cellule photo-électrique.
* JNM, Centre de Ressources Art Sensitif.
*/
int capteur1 = 0; // variable identifiant un port ana. 0 de la
carte
int LED1 = 13; // variable identifiant le port num. 13 de la
carte
int lumT = 0; // variable identifiant la valeur de la luminosité
du capteur 1
void setup()
{
pinMode(LED1, OUTPUT); // configure la broche 13 comme une
sortie
}
void loop()
{
lum1 = analogRead( capteur1); // lire la donnée capteur
digitalWrite(LED1, HIGH); // allumer la LED 1
delay(lum1); // attendre pendant la valeur donnée
par le capteur en millisecondes
digitalWrite(LED1, LOW); // éteindre la LED 1
delay(lum1); // attendre pendant la même valeur
}
```

Etape 2 : *Vérification* du programme

S'il n'y a pas d'erreurs on verra s'afficher: "***compilation terminée***", suivi de la taille du programme.



Etape 3 : *réinitialisation* de la carte



On peut sauver le fichier sur l'ordinateur, puis appuyer sur le bouton de ré-initialisation de la carte, ci-dessous.

Etape 4 : *Téléchargement* du programme

Et enfin, télécharger le programme sur l'Arduino: attention, vous avez 5 secondes après l'appui sur le bouton de ré-initialisation pour cliquer sur le bouton "Upload " !

Et voilà ! les deux petites LEDs TX RX sur la carte clignotent pendant le chargement, puis, quelques secondes plus tard, le programme se met en route…jusqu'à ce qu'on éteigne la carte...

III- Etape d'écriture d'un programme Arduino

1) Structure du programme

Un programme Arduino doit se structurer en 4 parties :

Partie 1 : commentaires
Cette première partie définit le programme. Il doit donner un résumé du fonctionnement du programme. Tout commentaire doit être précédé et se terminer par des balises spéciales.

```
/* Ce programme fait clignoter une LED branchée sur la broche 13
* et fait également clignoter la diode de test de la carte
*/
```

Partie 2 : Définition des variables.
Dans cette partie, vous devez définir quelles variables vous allez utiliser dans votre programme, ainsi que leur nature (binaire, entier, chaine de caractères, …)

```
int BrocheLED = 13; // Définition de la valeur 13 et du nom de la
broche à utiliser
```

Partie 3 : Configuration des entrées/sorties.
Dans cette partie vous définirez la nature des entrées/sorties matérielles utilisées (si telle ou telle broche est utilisé en entrée ou en sortie, de nature numérique (binaire : 0 ou 1) ou analogique.
Cette phase doit être précédée par la commande "***void setup()*** " suivie d'une accolade "***{***" et se terminer d'une accolade "***}***".

```
void setup()
{
pinMode(BrocheLED, OUTPUT); // configure BrocheLED comme
un sortie
}
```

Partie 4 : Programmation des interactions.
Cette partie définit ce que doit faire votre programme. C'est le programme à par entière. Elle doit être précédée de la commande "***void loop()***",d'une accolade "***{***" et se terminer d'une accolade "***}***".

```
void loop()
{
digitalWrite(BrocheLED, HIGH); // met la sortie num. à l'état haut
(led allumée)
delay(3000); // attente de 3 secondes
digitalWrite(BrocheLED, LOW); // met la sortie num. à l'état bas
(led éteinte)
delay(1000); // attente de 1 seconde
}
```

Un programme utilisateur Arduino est une suite d'instructions élémentaires sous forme textuelle, ligne par ligne.
La carte lit puis effectue les instructions les unes après les autres, dans l'ordre défini par les lignes de code.
Une fois la dernière ligne exécutée, la carte revient au début de la troisième phase et recommence sa lecture et son exécution des instructions successives. Et ainsi de suite.
Cette **boucle** se déroule des milliers de fois par seconde et anime la carte.

2) Les commentaires

La première partie du programme, doit impérativement débuter par des commentaires permettant de définir ce que doit faire le programme. Ceci permet de savoir en quoi consiste son fonctionnement, pourquoi pas sa date de création, sa version …etc

Mais les commentaires ne se limite pas à la première partie. Les commentaires doivent agrémenter votre programme afin de le faire comprendre par une personne tiers, essayant de le lire. Même pour vous, si vous devez un jour reprendre le programme et pourquoi pas le corriger ou l'améliorer, il est important de savoir à quoi servent telles ou telles lignes.

Si les commentaires doivent s'écrire sur plusieurs lignes, ils devront débuter par la balise suivante : " **/*** " , ensuite à chaque ligne supplémentaire débuter par " ***** " et lorsque votre commentaire est terminé, finir par la balise suivante : " */ " .

Si vous ne terminez pas par cette balise, tout ce que vous écrirez par la suite sera considéré comme commentaires, donc n'aura aucune action sur le programme.

Si les commentaires sont placés sur une seule ligne (par exemple après une instruction ou une définition de variable pour expliquer à quoi elle sert, ils devront simplement débutés par la balise " // ". Au saut de lignes suivant le commentaire sera terminé.

ATTENTION : Avant chaque changement de ligne, il faut terminer celle-ci par un point-virgule (avant les commentaires). Sauf pour les commentaires et les " ***void*** "

3) Définition des variables

Qu'est-ce qu'une variable ?

Déf. : Une variable est un espace de stockage nommé qui permet de stocker une valeur utilisable par la suite dans la boucle d'un programme.

Une variable peut aussi bien représenter des données lues ou envoyées sur un des ports analogiques ou numériques, une étape de calcul pour associer ou traiter des données, que le numéro 'physique' de ces entrées ou sorties sur la carte. Une "variable" n'est donc pas exclusivement un paramètre variant dans le programme.

*Pour composer un programme, il est nécessaire de définir toutes les **composantes** d'entrée et de sortie qui vont affecter le **montage matériel** et les **calculs à effectuer**. Chaque entrée et chaque sortie sera une variable.*

Définition de la nature de la variable.

Il existe plusieurs définitions des variables. Nous en verrons que 6, qui sont les plus utilisé.
La nature de la variable définit les limites de celle-ci, ainsi que la place nécessaire dans la mémoire de la carte Arduino. Ceci a de l'importance lorsque vous créez de gros programme impliquant beaucoup de variables qui prennent de la place en mémoire. Cette définition permet de limiter la place mémoire. De plus, il faut savoir que plus la variable prend de place, plus les opérations de calcul seront longues et inversement.

Il est donc nécessaire de définir les variables au plus juste en fonction de ce que le programme doit faire.

- **byte** : nombre de 0 à 255

- **char** : définit un caractère alphanumérique.
- **int** : entier décimal allant de -32 768 à 32 767. Va de pair avec **unsigned int** qui lui ira de 0 à 65 535 (il augmente les positifs et élimine les négatifs)
- **string** : suite de caractères alphanumérique. Pas de limite, sauf celle de la mémoire.
- **long**: entier décimal allant de -2 147 483 648 à 2 147 483 648. Va de pair avec **unsigned long** qui lui ira de 0 à 4 294 967 295 (il augmente les positifs et élimine les négatifs)
- **array :** définition d'un tableau de variable. A la différence des autres, lorsqu'on déclare un tableau, on ne note par " **array montableau** ". On notera :

```
int      montableau [10]  =      { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

Nature des variables utilisées dans le tableau — Nom du tableau — Nombre de variables composant le tableau — Valeurs inscrites dans le tableau

Pour ressortir une variable, par exemple "5", il suffira d'écrire :
montableau[6] et il donnera la valeur à la 6ème position du tableau, donc "5"

Pour chaque variable il faudra en plus de lui donner sa nature, il faudra lui donner une valeur d'initialisation.

Notation de valeur : vous avez 3 possibilités :
- **Noter en décimal, il suffit de noter le nombre**
- **Noter en binaire, il faudra noter juste avant le nombre un "B"**
- **Noter en hexadécimal, il suffira de noter juste avant le nombre "0x"**

Exemple : 0xA302 pour l'hexadécimal, ou B100110 pour le binaire.

4) Configuration des entrées/sorties.

Pour la configuration des entrées/sorties utilisées vous ne devez pas oublier "**void setup ( )**"

Dans cette configuration vous devrez définir comment doivent se comporter les différentes broches de l'ARDUINO.

Vous devrez définir les broches numériques, utilisant que du binaire, analogique servant soit aux convertisseurs ou à la PWM, ainsi qu'aux broches de communications.

a) Numérique : pinMode (Broche , état)

La première variable à mettre est le numéro de broche que l'on essaie de configurer

La deuxième variable est l'état de configuration, soit " **INPUT** " (entrée, il s'agit de broche recevant des informations), soit " **OUTPUT** " (sortie, il s'agit de broche envoyant des informations)

Exemple : pinMode (15, INPUT); // la broche 15 est configurée en entrée

Remarque : le nombre donné pour la broche, peut être remplacé par une variable. N'oubliez pas dans ce cas de configurer au préalable la variable.

b) Analogique.

Pour la configuration de broche analogique, seul une sortie analogique doit être configurée.

Vous devrez simplement ajouter **pinMode (numéro de broche, OUTPUT)**.

Pour les entrées analogiques, pas besoin de les définir, elles se mettront automatiquement en entrée lorsqu'on les solicitera.

ATTENTION : Certaines broches de l'ARDUINO peuvent être configurées soit en analogique soit en numérique. Si vous ne les configurées pas au départ, elles seront considérées comme entrées analogique.

c) Communication.

Certaines broches pourront être utilisées pour des type de communication, tel que liaison RS232, I2C ou autre. Ces broches ont des configurations particulières. Nous verrons certaines au fur et à mesure des TPs.

5) Programme principal.

void loop( ) : cette instruction définit le début du programme principal, celui qui tournera en permanence sur votre carte. La fonction loop permet justement de faire tourner en boucle votre programme.

Vous ne devrez pas oublier les accolades :

- En début de programme : " {"
- En fin de programme : " }"

Ces deux balises définissent le début et la fin de votre programme.

a) Lecture et écriture numérique et analogique :

- **digitalRead (Broche)**: cette instruction permet de lire la valeur d'une entrée. Comme cette valeur, pour être utilisable, doit être sauvegardée, nous allons utiliser une équation qui devra ressembler à ça :

variable = digitalRead (n° de broche)

Ainsi dans la variable sera enregistrée la valeur présente au n° de broche.

- **digitalWrite(Broche, état)**: cette instruction permet d'écrire sur une broche la valeur de l'état. Cet état sera soit HIGH (niveau haut : 1), soit LOW (niveau bas : 0).

Par exemple :

digitalWrite (15,HIGH); // la broche n°15 sera mise au niveau logique haut (1)

- **analogRead(Broche)**: cette instruction permet de lire une valeur analogique sur une des broches dédiées. Tout comme digitalRead, cette valeur doit être sauvegardée, il va donc falloir utiliser une équation de ce style :

variable = analogRead(0); //sauvegarder la valeur du convertisseur lié à la broche A0 (entrée analogique 0) dans la variable

ATTENTION : les convertisseurs de la carte, donneront des nombres binaire sur 10 bits, soit une valeur maximal de 1023. Ceci signifie que nous devrons définir au préalable la variable, en entier (int).

Chaque conversion se fera par rapport à une tension par défaut de 5V maxi. La valeur convertit sera une image de la tension qui va évoluer entre 0 et 5V, par tranche de 4,88 mV.

Par exemple si la tension sur A0 est de 2,5 V, la valeur obtenue après l'instruction analogRead sera $\frac{2,5}{0,00488} = 512$

- **analogWrite(Broche,valeur)** : cette instruction permet de créer une PWM :

Cette fonction appelée PWM (Pulse Width Modulation, Modulation à Largeur d'Impulsion), permet de générer un signal rectangulaire périodique de fréquence fixe (490 Hz, soit une période d'environ 2ms) mais de rapport cyclique variable (le temps au niveau haut évolue sans changer la période). Le rapport cyclique est réglé par la "valeur". Celle-ci peut être entre 0 et 255. Cette valeur divisée par 255, donnera le pourcentage de temps que prendra le niveau haut par rapport au niveau bas.

Exemple : analogWrite(5,128); // Un signal de 490 Hz sera présent sur la broche 5 de l'ARDUINO avec un rapport cyclique de $\frac{128}{255}$ = 50%. Soit un temps identique entre l'état haut et l'état bas.

Une fonction intéressante pour la gestion des ports et la fonction PORT (registre des ports).
Elle permet de manipuler en même temps les ports digitales.
Il existe plusieurs ports suivant le type de carte : PORTB, PORTC, PRTD
Il existe plusieurs registres :
DDRx : il permet d'indiquer les broches utilisées en sortie ou en entrée (DDRD=B11000000 les proches 6 et 7 sont définies en sortie)
PORTx : il permet d'indiquer l'état des sorties (PORTD=B10000000 met la borne 7 à HIGH et 6 à LOW)
PINx : il permet de lire l'état des bornes

b) Gestion du temps.

Il est souvent nécessaire d'effectuer des pauses dans le temps, de temporiser des programmes. Ceci est souvent utilisé dans la gestion de signaux numérique. La procédure avec l'ARDUINO est assez simple, il existe deux instructions permettant permettant d'effectuer des pauses :

- **delay (ms)** : cette instruction permet d'effectuer des pauses d'une durée en milli seconde inscrite entre parenthèses.

Exemple : delay(250); //une pause de 250 ms est effectuée

- **delayMicroseconds(µs)** : cette instruction permet d'effectuer des pauses d'une durée en micro seconde inscrite entre parenthèses.

Exemple : delay(150); //une pause de 150 µs est effectuée

- **millis( )**: cette instruction permet de connaître le temps passé depuis le lancement du programme. La valeur est donné en milliseconde. Ce nombre sera un nombre entier sans virgule. Il faudra sauvegarder cette valeur dans une variable. Attention cette variable devra être configuré en unsigned long (permettant de stocker un nombre de 4 294 967 295 maximum, soit ne durée maximum de 49 jours 17 h 2 min 47 s 295 ms) . Son utilisation est la suivante :

```
temps = millis( );      //le temps d'excecution en ms est stocké dans la variable temps
```

- **micros( )** : cette instruction permet de connaître le temps passé depuis le lancement du programme, mais en microseconde. Ce nombre sera un nombre entier sans virgule. Il faudra sauvegarder cette valeur dans une variable. Attention cette variable devra être configuré en unsigned long (permettant de stocker un nombre de 4 294 967 295 maximum, soit ne durée maximum de 1 h 11 min 34 s 967 ms 295 µs) . Son utilisation est la suivante :

```
temps = micros( );     //le temps d'exécution en µs est stocké dans la variable temps
```

c) Contrôle et condition.

Les conditions consistent à définir la réaction du logiciel par rapport à un résultat ou un événement.

Première condition : **If (condition sous forme d'équation)**

```
{
// Action si la condition est validée
}
```

La condition "If", signifie bien la traduction "si". En effet la fonction est de dire "si" la condition est bonne, alors nous effectuons une action, sinon le programme continu son chemin.

Chaque condition que vous mettrez sera obligatoirement sous forme d'équation.

ATTENTION : Les équation que vous mettrez ont une syntaxe bien particulière. Le langage ARDUINO distingue les syntaxes pour les conditions et les syntaxes pour les calculs.

Syntaxe pour les conditions :

x = = y (x est égal à y)
x != y (x n'est pas égal à y)
x < y (x est strictement inférieur à y)
x > y (x est strictement supérieur y)
x <= y (x est inférieur ou égal à y)
x >= y (x est supérieur ou égal à y)

Syntaxe pour les calculs :

= (opérateur d'affectation, signe égal des opérations), utiliser pour définir la valeur d'une variable.
+ (addition)
- (subtraction)
***** (multiplication)
/ (division)
% (modulo), le résultat de cette opération est le reste de la division. Exemple : 7 % 5 = 2, on effectue la division de 7 par 5 :

le reste est égal à 2. Autre exemple 30 % 4, donne 2 (7x4 = 28 et 30 - 28 = 2)

On peut ajouter plusieurs conditions, grâce à des opérateurs logiques, de type "ET", "OU", "OUI" …

par exemple :

Deux boutons, A et B, peuvent allumer une LED. Mais ces deux doux boutons doivent être appuyé pour que la LED s'allume.

Ceci peut se traduire par "SI A= = 1 ***ET*** B = = 1, alors la LEDs s'allume"
En langage ARDUINO, on écrira :

```
if (A= = 1 && B = = 1) // Si les boutons A et B sont appuyés
        {
        digitalWrite(LED,HIGH); // La LED s'allume
        }
```

ATTENTION: Les équation que vous mettrez ont une syntaxe bien particulière. Le langage ARDUINO distingue les syntaxes pour les conditions et les syntaxes pour les calculs.

Syntaxe pour les conditions :

- **&&** : signifie "ET"
- **|| (appuyez deux fois sur AltGr et la touche 6)**: signifie "OU"
- ! : signifie "NON" (l'inversion binaire)

Syntaxe pour les calculs :

- **&** : signifie le "ET"
- | **(appuyez sur AltGr et la touche 6)**: signifie "OU"
- ^: signifie XOR
- ~ : signifie "NON" (l'inversion binaire)
- << : signifie décalage à gauche bit à bit, il s'écrit "mot binaire" << "nombre de décalage"

Exemple :

```
int A = b00000101;
int B = A << 3; // le résultats sera 3 décalage à gauche, soit B = b00101000
```

- >> : signifie décalage à droite bit à bit, il s'écrit "mot binaire" << "nombre de décalage"

Exemple :

```
int A = b10100000;
int B = A >> 3; // le résultats sera 3 décalage à droite, soit B = b00010100
```

Deuxième condition :

```
If (condition sous forme d'équation)
        {
        // Action si la condition est validée
        }
else
        {
        // Action si la condition du if n'est pas validée
        }
```

Le "**else**" est une instruction que l'on peut combiner avec "**if**". Il permet une action si la condition du "**if**" n'est pas validé.

Troisième condition :

```
switch (variable)
        {
        case valeur1:
                // Action si la variable est égal à valeur1
                break ;
        case valeur2 :
                // Action si la variable est égal à valeur2
break ;
        … etc
        default :
        // Action si si la variable n'est égal à aucune des
valeurs
        }
```

Quatrième condition : **while (condition sous forme d'équation)**
{
// Action à faire tant que la condition est validée
}

Instruction de contrôle : **for (initialisation , condition, incrément ou décrément)**
{
// Action tant que la condition est validée
}

La traduction est la suivante :

- on initialise une variable à une valeur
- on installe une condition sous forme d'équation
- on incrémente ou décrémente la variable à chaque fois que l'action sera faite.
- on sort du "for" lorsque la condition n'est plus valide.

Exemple :

```
for (x = 0 , x < 100, x = x + 2)
                    {
                    // Action effectué tant que x < 100
                    } // on effectue l'opération x = x + 2 et on recommence l'action
```

ATTENTION : Il existe une manière simple de faire x + 1 ou x - 1, grâce à une syntaxe particulière. On appelle cela l'incrémentation ou la décrémentation.

Syntaxe pour les conditions :

- ***incrémentation : ++ . Deux actions possibles***
  - ***: 1-*** ***x = 2***
    y = ++ x ; // incrémente x (x = 3) et place la valeur de x dans y (y = 3)
  - ***2-*** ***x = 2;***
    y = x ++; // effectue l'opération y = x + 1 (y = 3), mais laisse l'ancienne valeur dans x (x = 2)
- ***incrémentation : - - . Deux actions pour cela :***
  - ***1-*** ***x = 2***
    y = - - x ; // décrémente x (x = 1) et place la valeur de x dans y (y = 1)
  - ***2-*** ***x = 2;***
    y = x - -; // effectue l'opération y = x - 1 (y = 1), mais laisse l'ancienne valeur dans x (x = 2)

6) Sous-programme, fonction ou macro.

Plusieurs noms sont donnés pour ce type de programme permettant d'être appelé à n'importe quel moment du programme principal et plusieurs fois si nécessaire.

L'appel de ce programme se fait de la manière suivante :

Dans un programme Arduino, il suffit de nommer le sous-programme, suivi de 2 parenthèses :

```
sousprog ( );
```

Pour construire le sous-programme :

```
void sousprog ( )
    {
    // instructions du sous-programme
    }
```

Fonction avec variable :

Exemple : calcul de la circonférence d'un cercle

```
int calculcirconference(int r){
int circ ;
circ=2*PI*r ;
return circ ;
}
```

ATTENTION : Il est de bon ton d'écrire les sous-programmes après le programme principal.

Port Arduino Méga :