

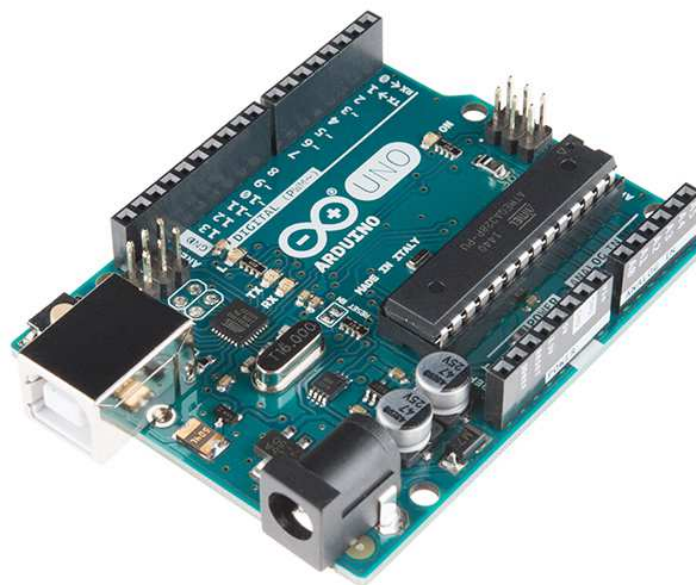


Athénée Royal d'Ans

Le langage C

Arduino

7 PEL



C. Baldewijns – F. Caprace – R. Radoux

2017-2018

Table des matières

Introduction	3
Le langage C.....	3
Plan du cours.....	3
Déroulement de l'année.....	3
Chapitre 1 : Le C, un langage évolué !	4
La programmation, qu'est-ce que c'est ?	4
L'Arduino UNO.....	7
Tester le fonctionnement de la carte	8
Le matériel et les composants.....	12
Chapitre 2 : Les exercices.....	15
TP1 : Les sorties digitales.....	15
TP2 : Clignotement de la LED	20
TP3 : Faire clignoter un groupe de LEDs.....	21
TP4 : Le chenillard.....	25
TP5 : Les entrées digitales	27
TP6 : Le capteur de niveau d'eau	31
TP7 : La photorésistance	33
TP8 : La sonde de température LM35	40
TP9 : Le moteur pas à pas.....	45
TP10 : Le servomoteur.....	48
Chapitre 3 : Le robot Timéo	00
Timéo, un drôle de robot	00
Chapitre 4 : Les exercices	00
TP1 :	00
Chapitre 5 : Compétences à atteindre en fin de cycle	00
En fin de période	00
Aux examens	00

Introduction

Le langage C

Le C est un langage incontournable qui en a inspiré beaucoup d'autres. Inventé dans les années 70, il est toujours d'actualité dans la programmation système et la robotique. Il est plutôt complexe, mais si vous le maîtrisez-vous aurez des bases de programmation très solides !

Dans ce cours, vous commencerez par découvrir le fonctionnement de la mémoire, des variables, des conditions et des boucles.

Plan du cours

Déroulement de l'année

Les périodes

1ère période : Chapitres 1

2ème période : Chapitre 2 et 3

3ème période : Chapitres 3

Objectif de ce cours

L'élève sera capable de réaliser le câblage et la programmation d'un Arduino Uno ainsi que la résolution de panne simple rapportée à ce dernier.

L'élève sera également apte à enseigner la programmation en C à des élèves de 1^{ère} et 2^{ème} année en énonçant les bases simplifiées de celle-ci ainsi.

Test du fonctionnement de la carte

TP1 : Les entrées numériques

Chapitre I : Le C, un langage évolué !

La programmation, qu'est-ce que c'est ?

Dans cette première partie, nous ferons nos premiers pas avec Arduino. Nous allons avant tout voir de quoi il s'agit exactement, essayer de comprendre comment cela fonctionne, puis installerons le matériel et le logiciel pour ensuite enchaîner sur l'apprentissage du langage de programmation nécessaire au bon fonctionnement de la carte Arduino.

La carte Arduino est une carte électronique qui ne sait rien faire sans qu'on lui dise quoi faire. Pourquoi ? Et bien c'est dû au fait qu'elle est **programmable**. Cela signifie qu'elle a besoin d'un **programme** pour fonctionner.

Un programme

Un programme est une liste d'instructions qui est exécutée par un système. Par exemple votre navigateur internet, est un programme. On peut analogiquement faire référence à une liste de course :

- Lait
- Pain
- Steak
- Epinards
- ...



Chaque élément de cette liste est une **instruction** qui vous dit : "Va chercher le lait" ou "Va chercher le pain", etc. Dans un programme le fonctionnement est similaire :

- Attendre que l'utilisateur rentre un site internet à consulter
- Rechercher sur internet la page demandée
- Afficher le résultat

Tel pourrait être le fonctionnement de votre navigateur internet. Il va attendre que vous lui demandiez quelque chose pour aller le chercher et ensuite vous le montrer. Eh bien, tout aussi simplement que ces deux cas, une carte électronique programmable suit une liste d'instructions pour effectuer les opérations demandées par le programme.

Des programmes, on peut en trouver de partout. Mais restons concentré sur Arduino. Le programme que nous allons mettre dans la carte Arduino, c'est nous qui allons le réaliser. Voici un exemple de programme C qui peut être envoyé directement dans la mémoire de l'Arduino.

```

/* 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, 2006, 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 lum1 = 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
}

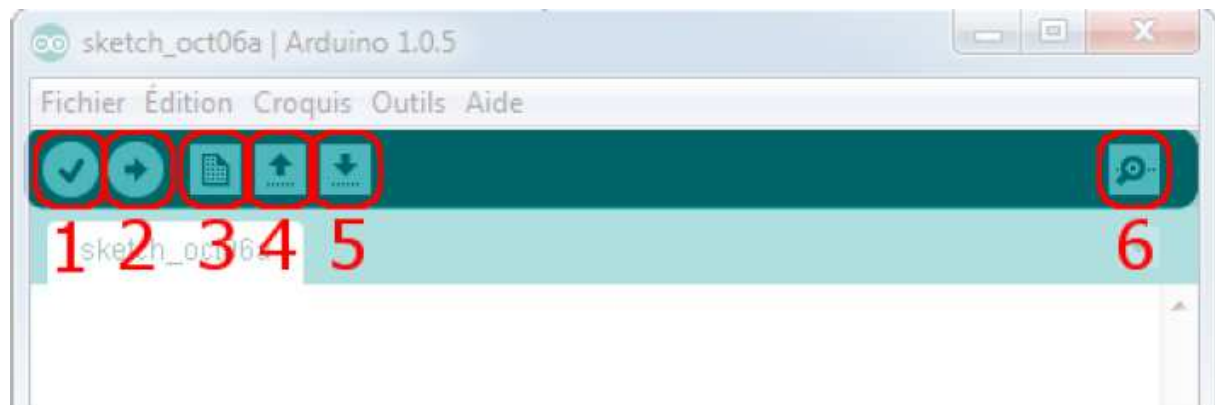
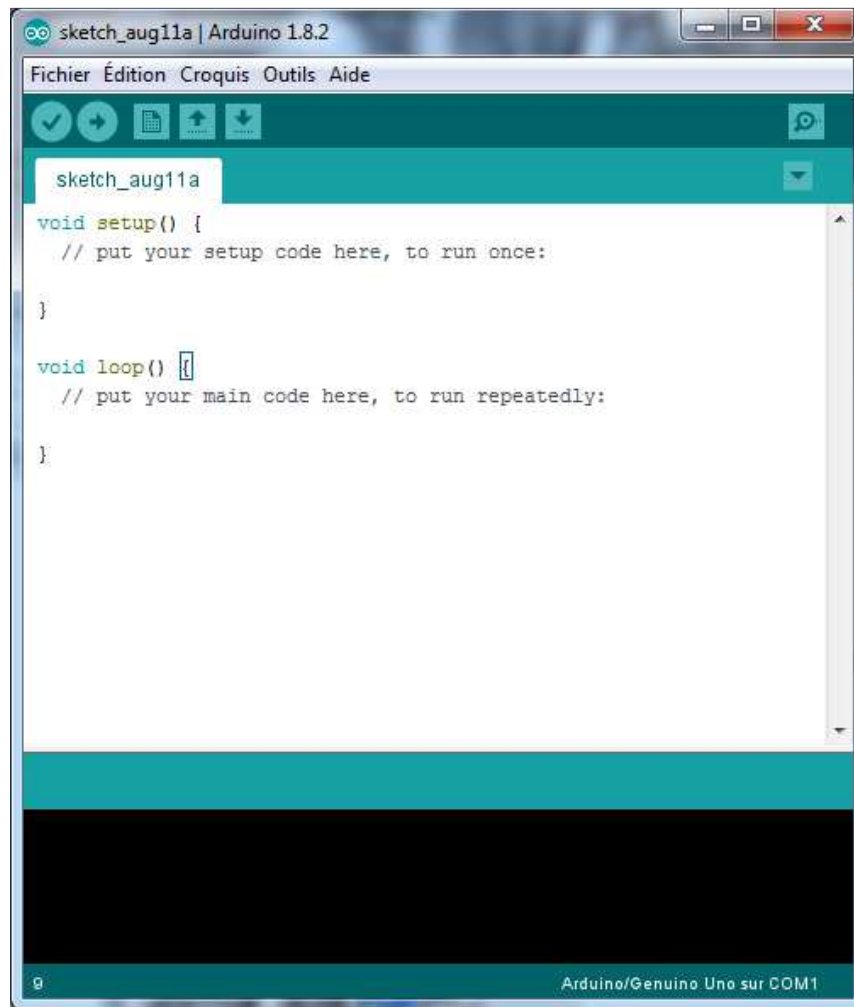
```

Ce langage est appelé langage C. Ce langage a besoin d'un logiciel pour être écrit et envoyé dans la carte Arduino, ce logiciel est totalement gratuit et disponible sur le NET à l'adresse suivante : <https://www.arduino.cc/en/Main/Software>



Le programme, une fois installé, à pour icône :

Cliquez deux fois dessus pour accéder aux lignes de code.



Bouton 1 : Ce bouton permet de vérifier le programme, il actionne un module qui cherche les erreurs dans votre programme

Bouton 2 : Charge (téléverse) le programme dans la carte Arduino

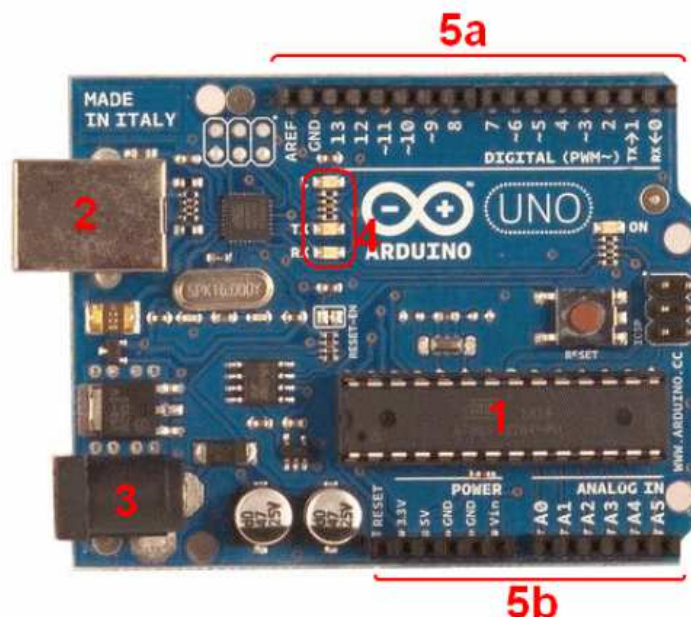
Bouton 3 : Crée un nouveau fichier

Bouton 4 : Ouvre un fichier

Bouton 5 : Enregistre le fichier

electricite.ara@gmail.com

L'arduino UNO



Exemples de carte électronique Arduino Uno

Constitution de la carte

Le microcontrôleur

Voilà le cerveau de notre carte (en **1**). C'est lui qui va recevoir le programme que vous aurez créé et qui va le stocker dans sa mémoire puis l'exécuter.

L'alimentation

Pour fonctionner, la carte a besoin d'une alimentation. Le microcontrôleur fonctionnant sous 5V, la carte peut être alimentée en 5V par le port USB (en **2**) ou bien par une alimentation externe (en **3**) qui est comprise entre 7V et 12V. Cette tension doit être continue et peut par exemple être fournie par une pile 9V. Un régulateur se charge ensuite de réduire la tension à 5V pour le bon fonctionnement de la carte

La visualisation

Les trois "points blancs" entourés en rouge (**4**) sont en fait des LED dont la taille est de l'ordre du millimètre. Ces LED servent à deux choses : elle clignote quelques secondes quand on branche la carte au PC. Les deux LED du bas du cadre : servent à le téléchargement du programme dans le microcontrôleur.

La connectique

La carte Arduino ne possédant pas de composants qui peuvent être utilisés pour un programme, mis à part la LED connectée à la broche 13 du microcontrôleur, il est nécessaire de les rajouter. Mais pour ce faire, il faut les connecter à la carte. C'est là qu'intervient la connectique de la carte (en **5a** et **5b**). Par exemple, on veut connecter une LED sur une sortie du microcontrôleur. Il suffit juste de la connecter, avec une résistance en série, à la carte, sur les fiches de connections de la carte. Cette connectique est importante et a un brochage qu'il faudra respecter.

Tester le fonctionnement de la carte

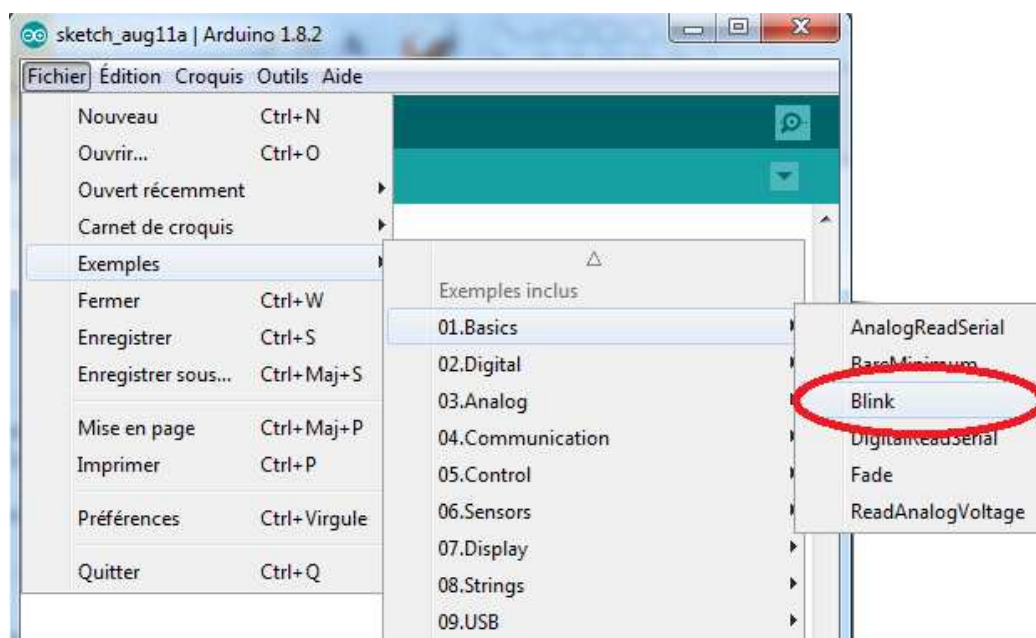


Carte connectée et alimentée

Avant de commencer à programmer la tête baissée, il faut, avant toutes choses, tester le bon fonctionnement de la carte. Car ce serait idiot de programmer la carte et chercher les erreurs dans le programme alors que le problème vient de la carte ! Nous allons tester notre matériel en chargeant un programme qui fonctionne dans la carte.

1ère étape : ouvrir un programme

Nous allons choisir un exemple tout simple qui consiste à faire clignoter une LED. Son nom est *Blink* et vous le trouverez dans la catégorie *Basics* :



Une fois que vous avez cliqué sur *Blink*, une nouvelle fenêtre va apparaître. Elle va contenir le programme *Blink*. Vous pouvez fermer l'ancienne fenêtre qui va ne nous servir plus à rien.

Voici un aperçu de la fenêtre de programmation. C'est CE programme tout fait que nous allons essayer d'envoyer à la carte électronique.



```

Blink $

/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

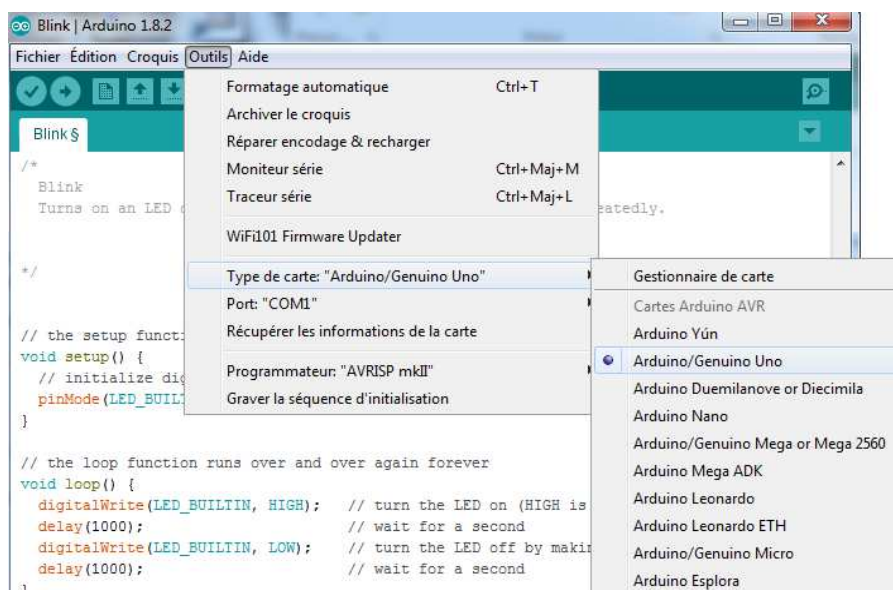
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}

```

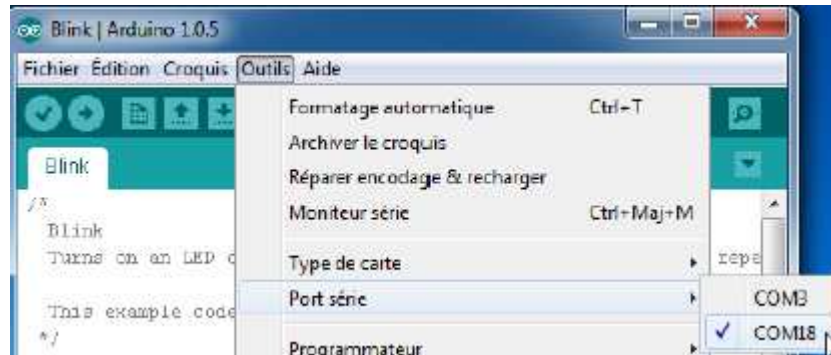
5 Arduino/Genuino Uno sur COM1

Mais avant d'envoyer le programme *Blink* vers la carte, il faut dire au logiciel quel est le nom de la carte et sur quel port elle est branchée. **Choisir la carte que l'on va programmer.** Ce n'est pas très compliqué, le nom de votre carte est indiqué sur elle. Pour nous, il s'agit de la carte "Uno".

Allez dans le menu "Tools" ("outils" en français) puis dans "Board" ("carte" en français). Vérifiez que c'est bien le nom "Arduin Uno" qui est coché. Si ce n'est pas le cas, cochez-le.



Ensuite Choisissez le port de connexion de la carte. Allez dans le menu *Tools*, puis *Serial port*. Là, vous choisissez le port COMX, X étant le numéro du port qui est affiché. Ne choisissez pas COM1 car il n'est quasiment jamais connecté à la carte. Dans mon cas, il s'agit de COM5 :

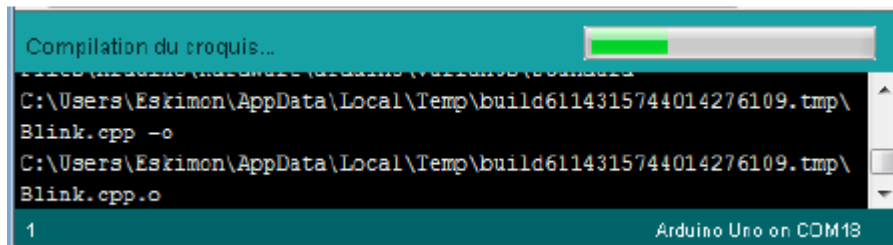


Pour la dernière étape, il va falloir envoyer le programme dans la carte. Pour ce faire, il suffit de cliquer sur le bouton *Téléverser*, en blanc sur la photo :

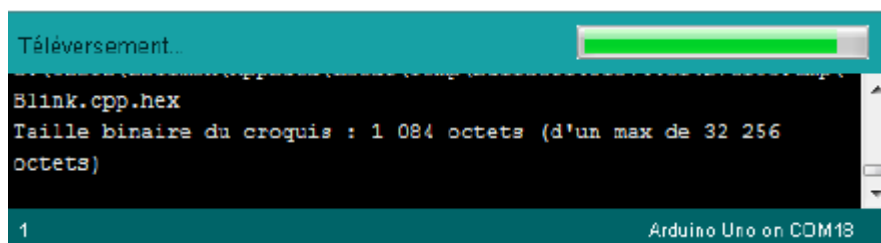


Les messages d'informations

Vous verrez tout d'abord le message "Compilation du croquis en cours..." pour vous informer que le programme est en train d'être compilé en langage machine avant d'être envoyé. Ensuite vous aurez ceci :

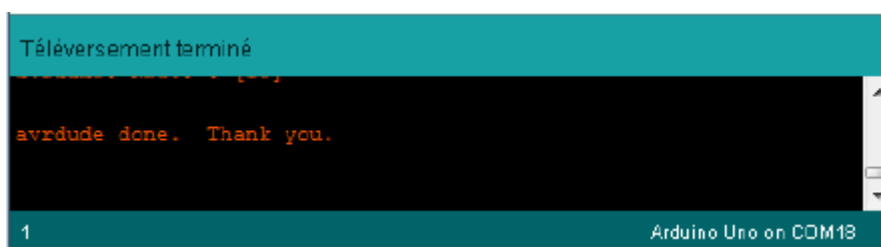


En bas dans l'image, vous voyez le texte : "Téléversement...", cela signifie que le logiciel est en train d'envoyer le programme dans la carte. Une fois qu'il a fini, il affiche un autre message :



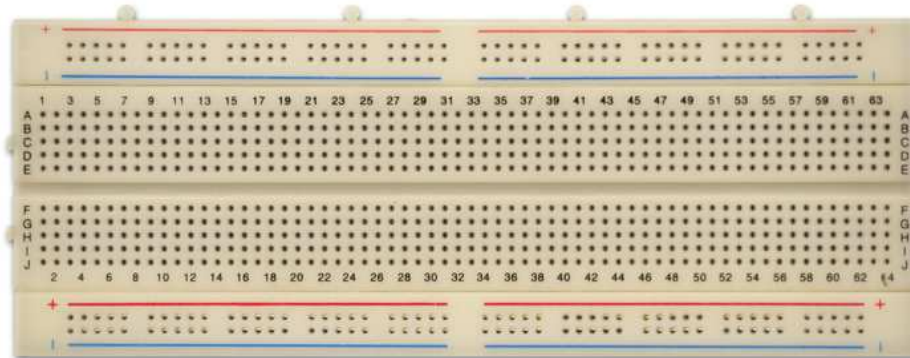
Le message afficher : "Téléversement terminé" signale que le programme à bien été chargé dans la carte. Si votre matériel fonctionne, vous devriez avoir une LED sur la carte qui clignote : Si vous n'obtenez pas ce message mais plutôt un truc en rouge, pas d'inquiétude, le matériel n'est pas forcément défectueux ! En effet, plusieurs erreurs sont possibles :

- l'IDE recompile avant d'envoyer le code, vérifier la présence d'erreur
- La voie série est peut-être mal choisi, vérifier les branchements et le choix de la voie série
- l'IDE est codé en JAVA, il peut être capricieux et bugger de temps en temps (surtout avec la voie série...) : réessayez l'envoi!



Le matériel et les composants

Principe de la breadboard

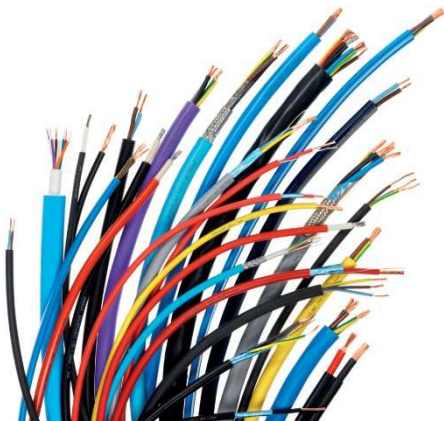


Certes la plaque est pleine de trous, mais pas de manière innocente ! En effet, la plupart d'entre eux sont reliés. Voici un petit schéma rapide qui va aider à la compréhension.

Par convention, le noir représente la masse et le rouge est l'alimentation (+5V, +12V, 5V... ce que vous voulez y amener). Habituellement tous les trous d'une même **ligne** sont reliés sur cette zone. Ainsi, vous avez une ligne d'alimentation parcourant tout le long de la carte. Ensuite, on peut voir des zones en bleu. Ces zones sont reliées entre elles par **colonne**. Ainsi, tous les trous sur une même colonne sont reliés entre eux.

En revanche, chaque colonne est distincte. En faisant chevaucher des composants sur plusieurs colonnes vous pouvez les connecter entre eux. Dernier point, vous pouvez remarquer un espace coupant la carte en deux de manière symétrique. Cette espace coupe aussi la liaison des colonnes. Ainsi, sur le dessin ci-dessus on peut voir que chaque colonne possède 5 trous reliés entre eux.

Les fils de liaisons



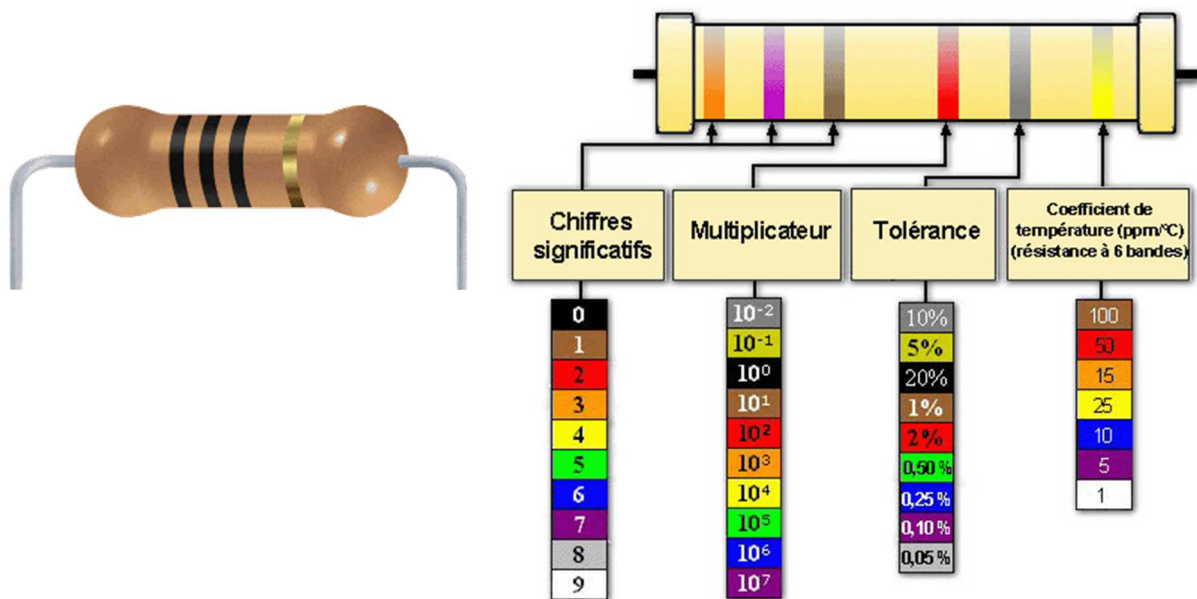
Pour faire la jonction entre les composants électroniques et la breadboard, nous utiliserons **des fils** de couleurs assez fin.

Les boutons poussoirs

Utiliser pour leurs contacts NO ou NF, ces boutons poussoirs vont nous servir à faire passer des courants électriques dans nos composants électroniques grâce à une action manuelle sur l'extrémité de ceux-ci.



Les résistances



En parallèle :

Pour 2 résistances :

$$R_{eq} = \frac{R_1 \cdot R_2}{R_1 + R_2}$$

Pour n résistances :

$$\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \dots + \frac{1}{R_n}$$

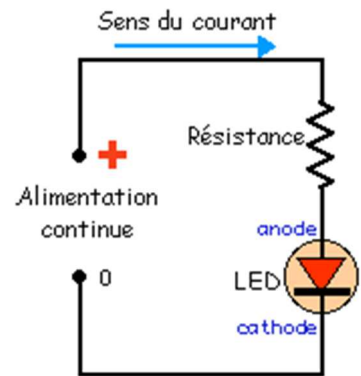
$$U = U_{R1} = U_{R2} = U_{R3} \quad \text{et} \quad I_{tot} = I_{R1} + I_{R2} + I_{R3}$$

En série :

Pour n résistances :

$$R_{eq} = R_1 + R_2 + R_3 + \dots + R_n$$

La LED



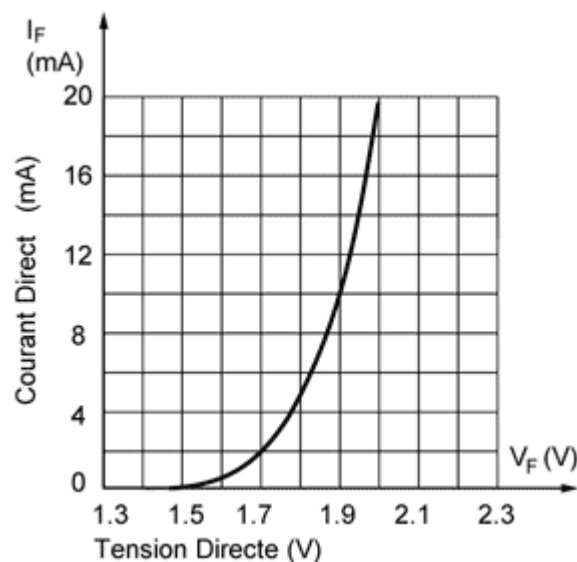
Le mot LED est l'acronyme de Light Emitting Diode (Diode Electroluminescente en français). Le symbole de la LED ressemble à celui de la diode mais on y a ajouté deux flèches sortantes pour représenter le rayonnement lumineux émis.

Cette diode émet de la lumière quand elle polarisée en direct.



En pratique, le constructeur préconise 10 à 20 mA. Le courant traversant la LED détermine l'intensité lumineuse émise. Le courant MAXIMUM qui doit passer dans la Led ne peut pas être supérieur à 25 mA.

La tension de conduction de la diode LED est d'environ 1,5 volts à 2 V.



Les résistances photo-électriques

La photorésistance n'a pas de sens de branchement. Sa valeur en ohms dépend de l'éclairement qu'elle reçoit. On la désigne aussi par LDR (Light Dependent Resistor = résistance dépendant de la lumière). La principale utilisation de la photorésistance est la mesure de l'intensité lumineuse.



Chapitre 2 : Les exercices

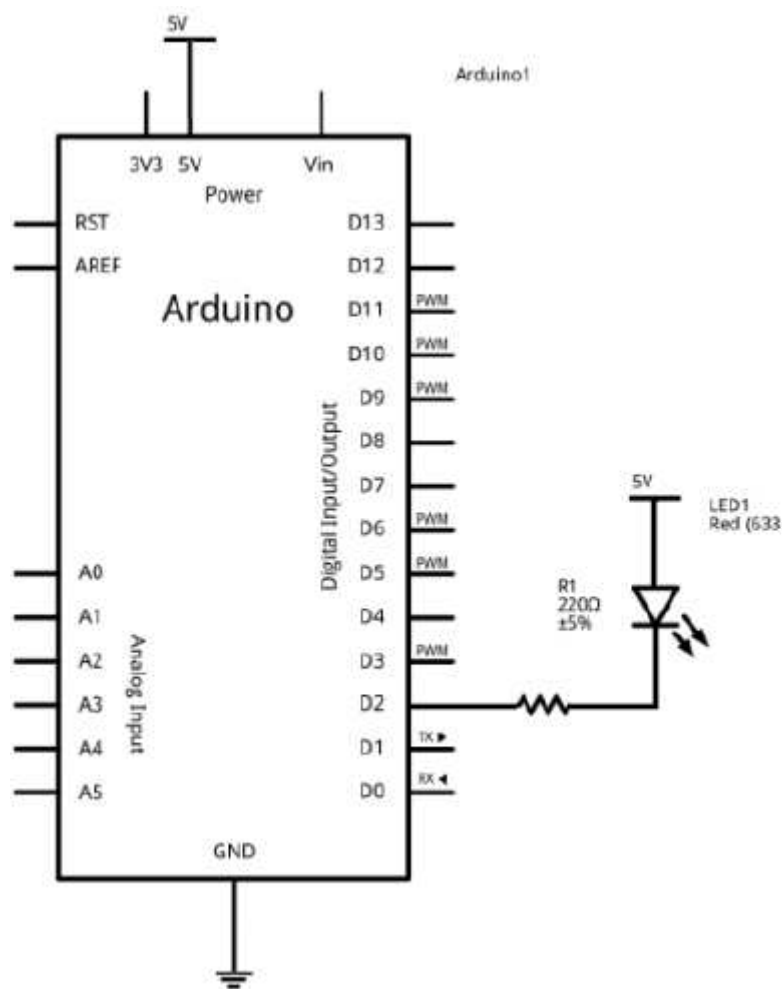
TP1 - Les sorties digitales

But : câbler 1 LED mises en série avec sa résistance de protection (220 Ω)

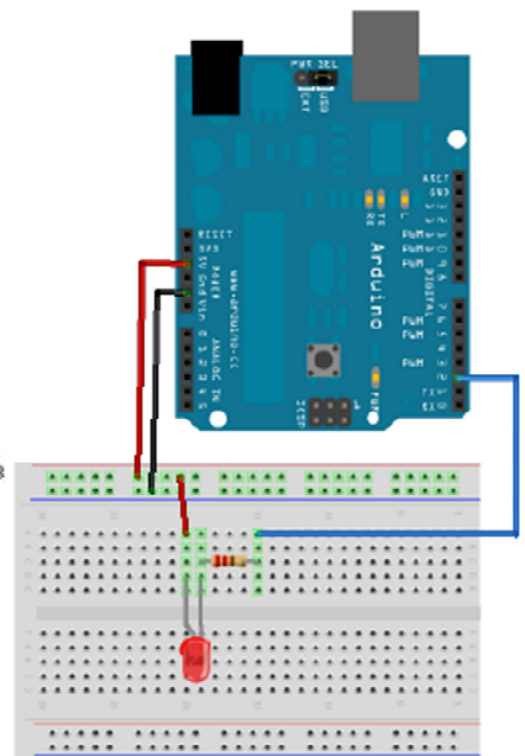
Avec le brochage de la carte Arduino, vous devrez connecter la plus grande patte au +5V (broche 5V).

La plus petite patte étant reliée à la résistance, elle-même reliée à la broche numéro 2 de la carte.

Schéma de liaison :

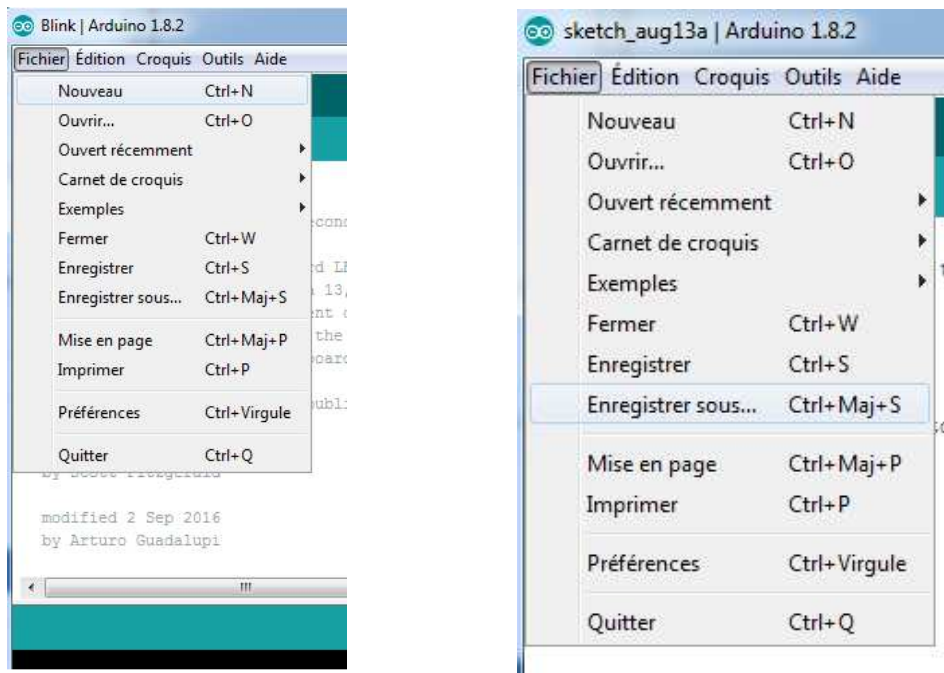


Raccordement à la carte :

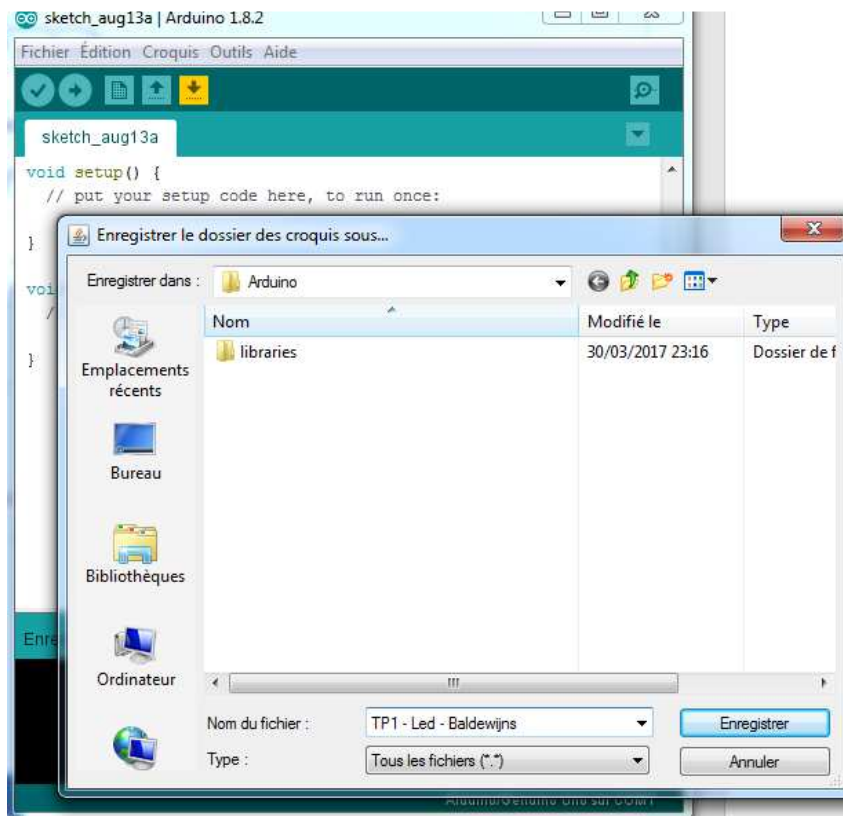


Programmation de la LED

Créer un nouveau projet dans le logiciel Arduino puis enregistrer sous...

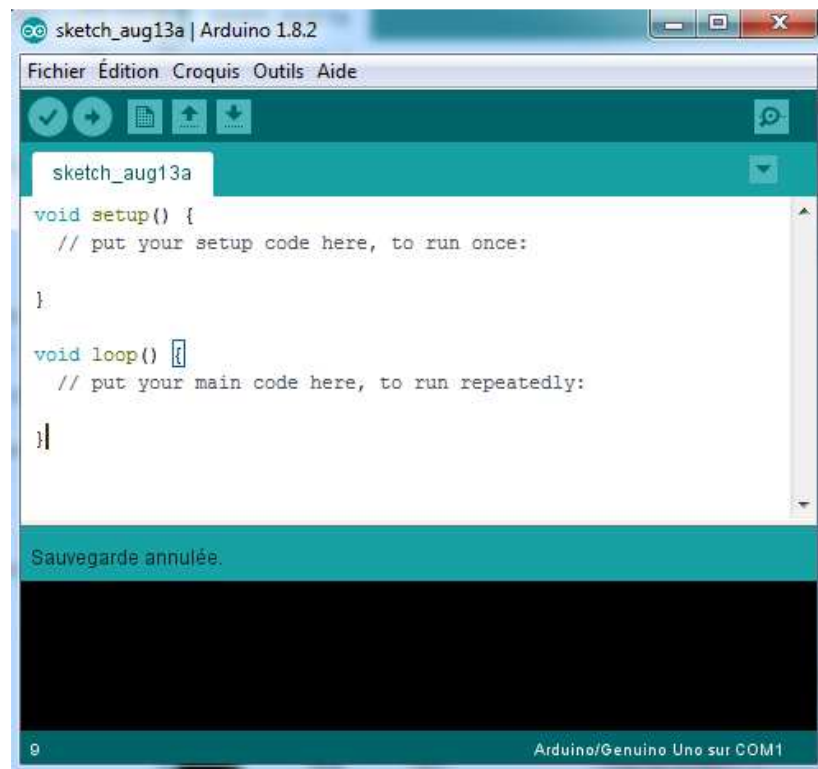


Et enregistrez le nom du TP et votre Nom comme dans l'exemple ci-dessous.



electricite.ara@gmail.com

Le nouveau programme est composé de deux parties, une partie void setup() où on placera les initialisation de broches en entrées ou en sorties et une partie void loop() qui nous permettra de créer une boucle continue.



1 ère étape :

Pour commencer, il faut définir les broches du microcontrôleur que nous allons utiliser. La première étape est de créer une variable définissant la broche utilisée, ensuite, définir si la broche utilisée doit être une entrée du microcontrôleur ou une sortie.

Premièrement, ajoutons la broche utilisée du microcontrôleur ou nous avons placé notre LED:

```
const int led_rouge = 2; //définition de la broche 2 de la carte en tant que variable constante
```

Le terme `int` correspond à un type de variable. En définissant une variable de ce type, elle peut stocker un nombre allant de -2147483648 à +2147483647 !

Nous sommes donc en présence d'une variable, nommée *led_rouge*, qui est en fait une constante, qui peut prendre une valeur allant de -2147483648 à +2147483647. Dans notre cas, cette variable, pardon constante, est assignée à 2. Le chiffre 2.

Lorsque votre code sera compilé, le microcontrôleur saura ainsi que sur sa broche numéro 2, il y a un élément connecté.

Deuxièmement il faut maintenant dire si cette broche est une **entrée** ou une **sortie**. Oui, car le microcontrôleur a la capacité d'utiliser certaines de ses broches en entrée ou en sortie.

En effet, il suffit simplement d'interchanger UNE ligne de code pour dire qu'il faut utiliser une broche en entrée (récupération de donnée) ou en sortie (envoi de donnée).

Cette ligne de code justement, parlons-en ! Elle doit se trouver dans la fonction **setup()**. Dans la référence, ce dont nous avons besoin se trouve dans la catégorie **Functions**, puis dans **Digital I/O**. I/O pour Input/Output, ce qui signifie : Entrée/Sortie. La fonction se trouve être **pinMode()**. Pour utiliser cette fonction, il faut lui envoyer deux paramètres :

- Le nom de la variable que l'on a défini à la broche
- Le type de broche que cela va être (entrée ou sortie)

```
//fonction d'initialisation de la carte
void setup()
{
//initialisation de la broche 2 comme étant une sortie
pinMode(led_rouge, OUTPUT);
}
```

Ce code va donc définir la led_rouge (qui est la broche numéro 2 du microcontrôleur) en sortie, car **OUTPUT** signifie en français : *sortie*.

Maintenant, tout est prêt pour créer notre programme. Voici le code quasiment complet :

```
sketch_aug13a $
//définition de la broche 2 de la carte en tant que variable
const int led_rouge = 2;
//fonction d'initialisation de la carte
void setup()
{
//initialisation de la broche 2 comme étant une sortie
pinMode(led_rouge, OUTPUT);
}
//fonction principale, elle se répète (s'exécute) à l'infini
void loop()
{
//contenu de votre programme
}

Sauvegarde annulée.
```

2ème étape :

Cette deuxième étape consiste à créer le contenu de notre programme. Celui qui va aller remplacer le commentaire dans la fonction **loop()**, pour réaliser notre objectif : allumer la LED !

On cherche une fonction donc dans la catégorie **Functions** de la référence. Cette fonction se trouve plus précisément dans **Digital I/O**. Tiens, il y a une fonction suspecte qui se prénomme **digitalWrite()**.

En français, cela signifie “écriture numérique”. C’est donc l’écriture d’un état logique (0 ou 1). Quel se trouve être la première phrase dans la description de cette fonction ? Celle-ci : “Write a HIGH or a LOW value to a digital pin”. D’après notre niveau bilingue, on peut traduire par : *Ecriture d’une valeur HAUTE ou une valeur BASSE sur une sortie numérique*. Bingo ! C’est ce que l’on recherchait !

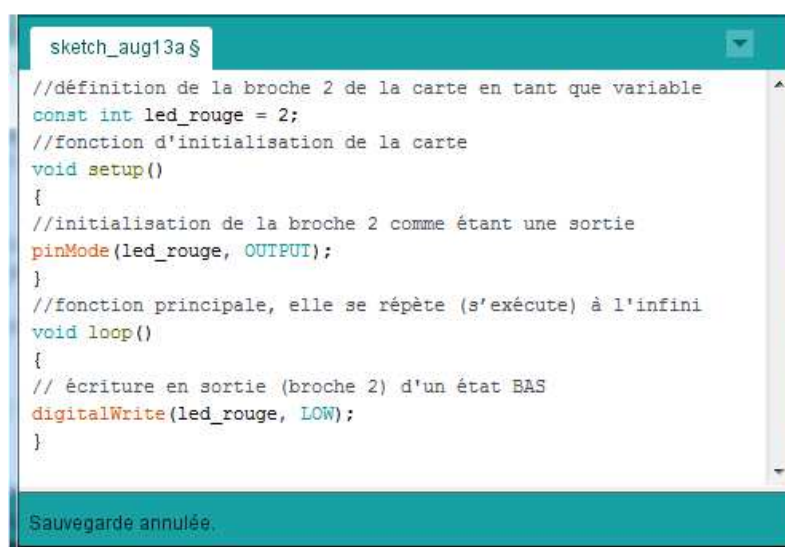
Ce signifie quoi “valeur HAUTE ou valeur BASSE” ?

En électronique numérique, un niveau haut correspondra à une tension de +5V et un niveau dit bas sera une tension de 0V (généralement la masse). Sauf qu’on a connecté la LED au pôle positif de l’alimentation, donc pour qu’elle s’allume, il faut qu’elle soit reliée au 0V. Par conséquent, on doit mettre un état bas sur la broche du microcontrôleur. Ainsi, la différence de potentiel aux bornes de la LED permettra à celle-ci de s’allumer.

Voyons un peu le fonctionnement de **digitalWrite()** en regardant dans sa syntaxe. Elle requiert deux paramètres. Le nom de la broche que l’on veut mettre à un état logique et la valeur de cet état logique. Nous allons donc écrire le code qui suit, d’après cette syntaxe :

digitalWrite(led_rouge, LOW); // écriture en sortie (broche 2) d'un état BAS

Si on teste le code entier : On voit s’éclairer la LED !!! C’est fantastique !



```

sketch_aug13a $
//définition de la broche 2 de la carte en tant que variable
const int led_rouge = 2;
//fonction d'initialisation de la carte
void setup()
{
  //initialisation de la broche 2 comme étant une sortie
  pinMode(led_rouge, OUTPUT);
}
//fonction principale, elle se répète (s'exécute) à l'infini
void loop()
{
  // écriture en sortie (broche 2) d'un état BAS
  digitalWrite(led_rouge, LOW);
}
Sauvegarde annulée.

```

TP2 – Clignotement de la LED

But : câbler 1 LED mises en série avec sa résistance de protection (220Ω) et la faire clignoter.

On va utiliser : **delay()** Petite description de la fonction, elle va servir à mettre en pause le programme pendant un temps prédéterminé.

Utiliser la commande

La fonction admet un paramètre qui est le temps pendant lequel on veut mettre en pause le programme. Ce temps doit être donné en millisecondes. C'est à dire que si vous voulez arrêter le programme pendant 1 seconde, il va falloir donner à la fonction ce même temps, écrit en millisecondes, soit 1000ms. Le code est simple à utiliser, il est le suivant :

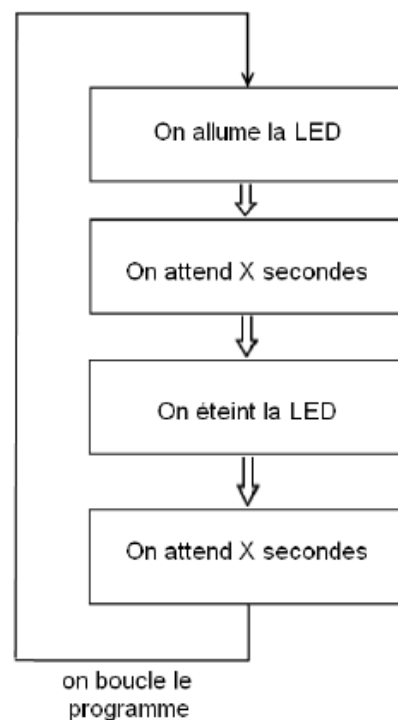
```
// on fait une pause du programme pendant 1000ms, soit 1 seconde
delay(1000);
```

Rien de plus simple donc. Pour 20 secondes de pause, il aurait fallu écrire :

```
delay(20000);
```

Mettre en pratique : faire clignoter une LED

Du coup, si on veut faire clignoter notre LED, il va falloir utiliser cette fonction. Voyons un peu le schéma de principe du clignotement d'une LED :



electricite.ara@gmail.com

Vous le voyez, la LED s'allume. Puis, on fait intervenir la fonction `delay()`, qui va mettre le programme en pause pendant un certain temps. Ensuite, on éteint la LED. On met en pause le programme. Puis on revient au début du programme. On recommence et ainsi de suite. C'est cette somme de commande, qui forme le processus qui fait clignoter la LED.



```

sketch_aug13a | Arduino 1.8.2
Fichier Édition Croquis Outils Aide

sketch_aug13a $
//définition de la broche 2 de la carte en tant que variable
const int led_rouge = 2;
//fonction d'initialisation de la carte
void setup()
{
  //initialisation de la broche 2 comme étant une sortie
  pinMode(led_rouge, OUTPUT);
}
void loop()
{
  // allume la LED
  digitalWrite(led_rouge, LOW);
  // fait une pause de 600 ms
  delay(600);
  // éteint la LED
  digitalWrite(led_rouge, HIGH);
  // fait une pause de 40 ms
  delay(40);
}

Sauvegarde annulée.

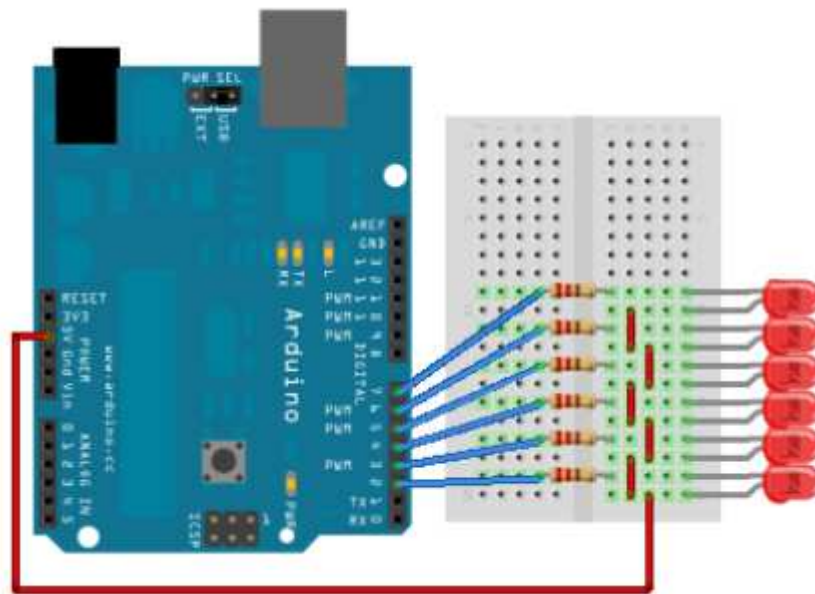
```

TP3 – Faire clignoter un groupe de LED

Vous avouerez facilement que ce n'était pas bien difficile d'arriver jusque-là. Alors, à présent, accentuons la difficulté. Notre but : faire clignoter un groupe de LED.

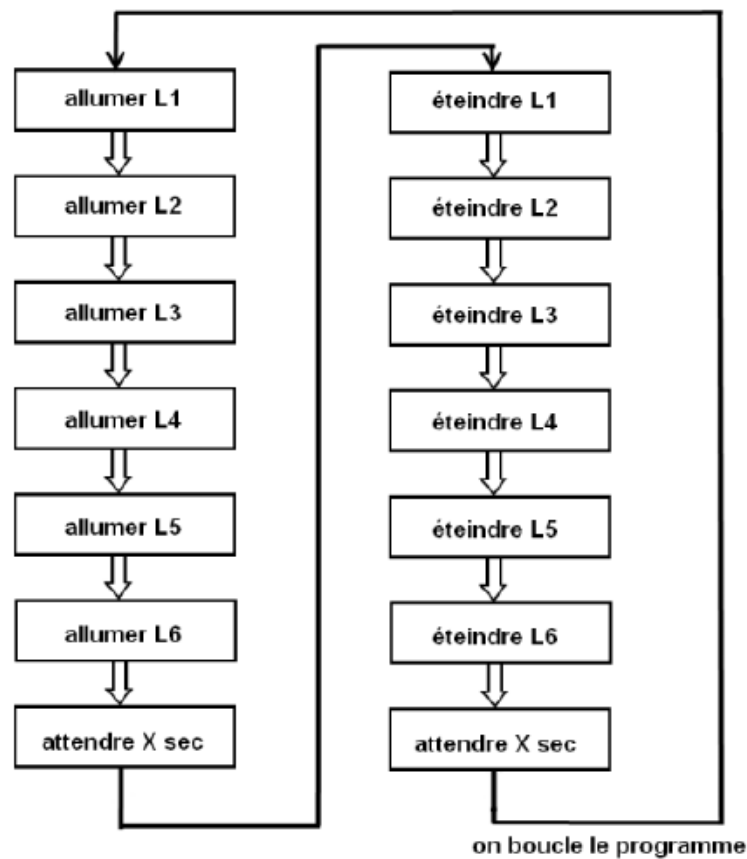
Ce groupe de LED sera composé de six LED, nommées L1, L2, L3, L4, L5 et L6. Vous aurez par conséquent besoin d'un nombre semblable de résistances.

Le schéma de la réalisation :



Programmation des LEDs :

Le programme est un peu plus long que le précédent, car il ne s'agit plus d'allumer 1 seule LED, mais 6 ! Voilà l'organigramme que va suivre notre programme :



Traduction des six premières instructions :

```
digitalWrite(L1, LOW); //notez que le nom de la broche à changé
digitalWrite(L2, LOW); //et ce pour toutes les LED connectées
digitalWrite(L3, LOW); //au microcontrôleur
digitalWrite(L4, LOW);
digitalWrite(L5, LOW);
digitalWrite(L6, LOW);
```

Ensuite, on attend 1,5 seconde :

```
delay(1500);
```

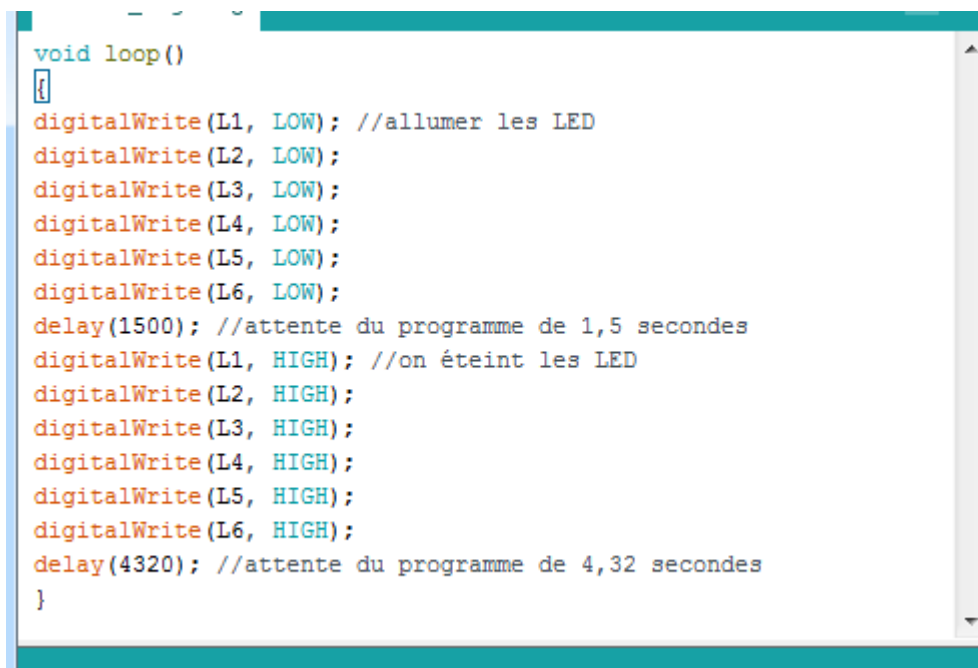
Puis on traduit les six autres instructions :

```
digitalWrite(L1, HIGH); //on éteint les LED
digitalWrite(L2, HIGH);
digitalWrite(L3, HIGH);
digitalWrite(L4, HIGH);
digitalWrite(L5, HIGH);
digitalWrite(L6, HIGH);
```

Enfin, la dernière ligne de code, disons que nous attendrons 4,32 secondes :

```
delay(4320);
```

Tous ces bouts de code sont à mettre à la suite et dans la fonction `loop()` pour qu'ils se répètent.



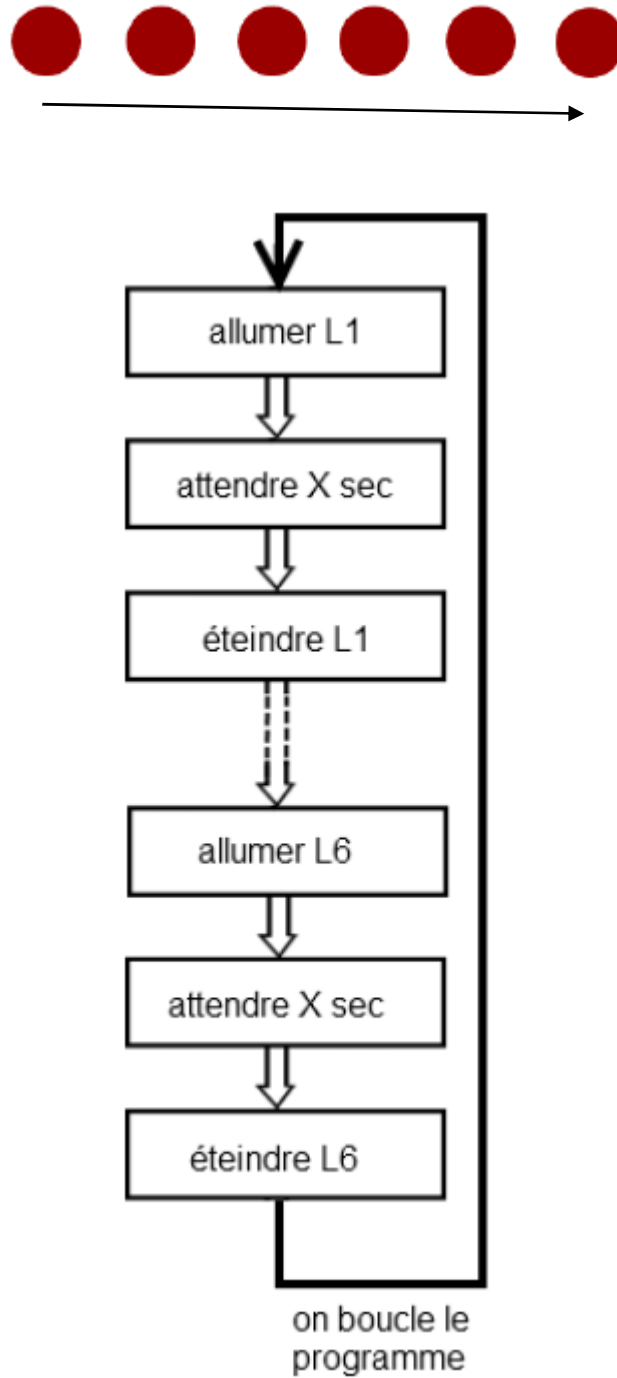
```
void loop()
{
    digitalWrite(L1, LOW); //allumer les LED
    digitalWrite(L2, LOW);
    digitalWrite(L3, LOW);
    digitalWrite(L4, LOW);
    digitalWrite(L5, LOW);
    digitalWrite(L6, LOW);
    delay(1500); //attente du programme de 1,5 secondes
    digitalWrite(L1, HIGH); //on éteint les LED
    digitalWrite(L2, HIGH);
    digitalWrite(L3, HIGH);
    digitalWrite(L4, HIGH);
    digitalWrite(L5, HIGH);
    digitalWrite(L6, HIGH);
    delay(4320); //attente du programme de 4,32 secondes
}
```


Le programme final : il n'est certes pas très beau, mais il fonctionne :



```
sketch_aug15a$  
  
const int L1 = 2; //broche 2 du microcontrôleur  
se nomme maintenant : L1  
const int L2 = 3; //broche 3 du microcontrôleur  
se nomme maintenant : L2  
const int L3 = 4; // ...  
const int L4 = 5;  
const int L5 = 6;  
const int L6 = 7;  
void setup()  
{  
  pinMode(L1, OUTPUT); //L1 est une broche de sortie  
  pinMode(L2, OUTPUT); //L2 est une broche de sortie  
  pinMode(L3, OUTPUT); // ...  
  pinMode(L4, OUTPUT);  
  pinMode(L5, OUTPUT);  
  pinMode(L6, OUTPUT);  
}  
void loop()  
{  
  //allumer les LED  
  digitalWrite(L1, LOW);  
  digitalWrite(L2, LOW);  
  digitalWrite(L3, LOW);  
  digitalWrite(L4, LOW);  
  digitalWrite(L5, LOW);  
  digitalWrite(L6, LOW);  
  //attente du programme de 1,5 secondes  
  delay(1500);  
  //on éteint les LED  
  digitalWrite(L1, HIGH);  
  digitalWrite(L2, HIGH);  
  digitalWrite(L3, HIGH);  
  digitalWrite(L4, HIGH);  
  digitalWrite(L5, HIGH);  
  digitalWrite(L6, HIGH);  
  //attente du programme de 4,32 secondes  
  delay(4320);  
}
```

TP4 – Le chenillard



Voilà donc ce qu'est un chenillard. Chaque LED s'allume alternativement et dans l'ordre chronologique. De la gauche vers la droite ou l'inverse, c'est au choix.

Normalement, la conception du programme ne devrait pas vous poser de problèmes. Il suffit en effet de récupérer le code du programme précédent (“allumer un groupe de LED”) et de le modifier en fonction de notre besoin. Ce code, je vous le donne, avec les commentaires :

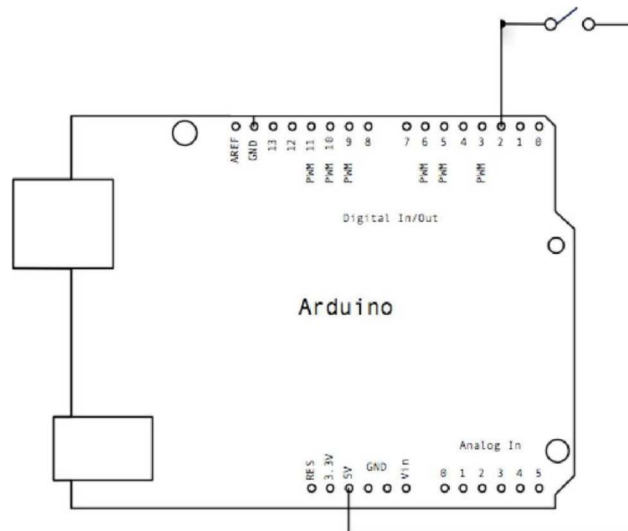
```

sketch_aug15a $
const int L1 = 2; //broche 2 du microcontrôleur se nomme maintenant : L1
const int L2 = 3; //broche 3 du microcontrôleur se nomme maintenant : L2
const int L3 = 4; // ...
const int L4 = 5;
const int L5 = 6;
const int L6 = 7;
void setup()
{
  pinMode(L1, OUTPUT); //L1 est une broche de sortie
  pinMode(L2, OUTPUT); //L2 est une broche de sortie
  pinMode(L3, OUTPUT); // ...
  pinMode(L4, OUTPUT);
  pinMode(L5, OUTPUT);
  pinMode(L6, OUTPUT);
}
// on change simplement l'intérieur de la boucle pour atteindre notre objectif
void loop() //la fonction loop() exécute le code qui suit en le répétant en boucle
{
  digitalWrite(L1, LOW); //allumer L1
  delay(1000); //attendre 1 seconde
  digitalWrite(L1, HIGH); //on éteint L1
  digitalWrite(L2, LOW); //on allume L2 en même temps que l'on éteint L1
  delay(1000); //on attend 1 seconde
  digitalWrite(L2, HIGH); //on éteint L2 et
  digitalWrite(L3, LOW); //on allume immédiatement L3
  delay(1000); // ...
  digitalWrite(L3, HIGH);
  digitalWrite(L4, LOW);
  delay(1000);
  digitalWrite(L4, HIGH);
  digitalWrite(L5, LOW);
  delay(1000);
  digitalWrite(L5, HIGH);
  digitalWrite(L6, LOW);
  delay(1000);
  digitalWrite(L6, HIGH);
}

```

TP5 - Les entrées digitales

Lorsque l'on utilise les entrées numériques de l'Arduino, celles que l'on peut lire par l'instruction `digitalRead(...)`, certains principes de base doivent être pris en compte. Considérons la connexion d'un bouton poussoir à une entrée numérique de l'Arduino. On pourrait penser qu'il suffit de connecter un poussoir comme suit :

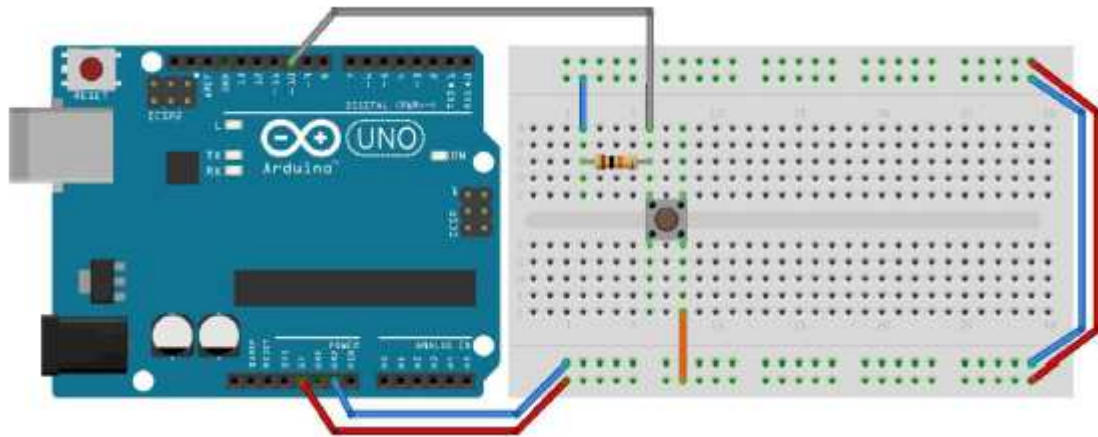


Lorsque le bouton est pressé, l'entrée est à 5 V, mais lorsque le bouton n'est pas pressé, elle n'est pas définie car elle est "en l'air".

Les entrées du microcontrôleur équipant l'Arduino sont très sensibles et elles réagissent à la tension qui leur est appliquée. En conséquence, l'entrée laissée en l'air peut avoir n'importe quelle valeur comprise entre 0 et 5 V et être interprétée comme un LOW ou un HIGH par l'instruction `digitalRead(...)`.

Résistance pull down

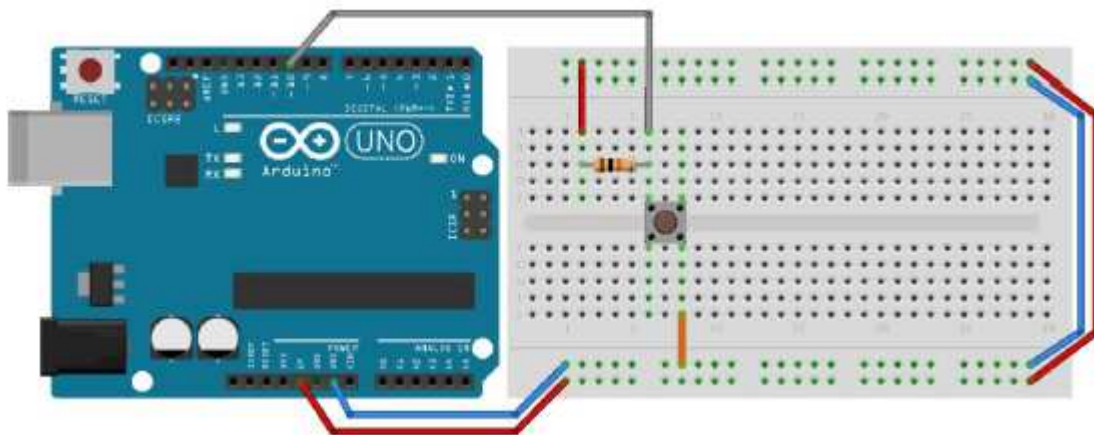
Pour fixer la tension lorsque le bouton n'est pas pressé, on insère une résistance pull down (résistance de tirage vers le bas) entre l'entrée et la masse (GND) dont le rôle est d'assurer que l'entrée est à 0 V lorsque le poussoir n'est pas pressé.



Pour une alimentation de 5 V, une bonne valeur pour R est d'au moins 4.7 Kilo-Ohm.

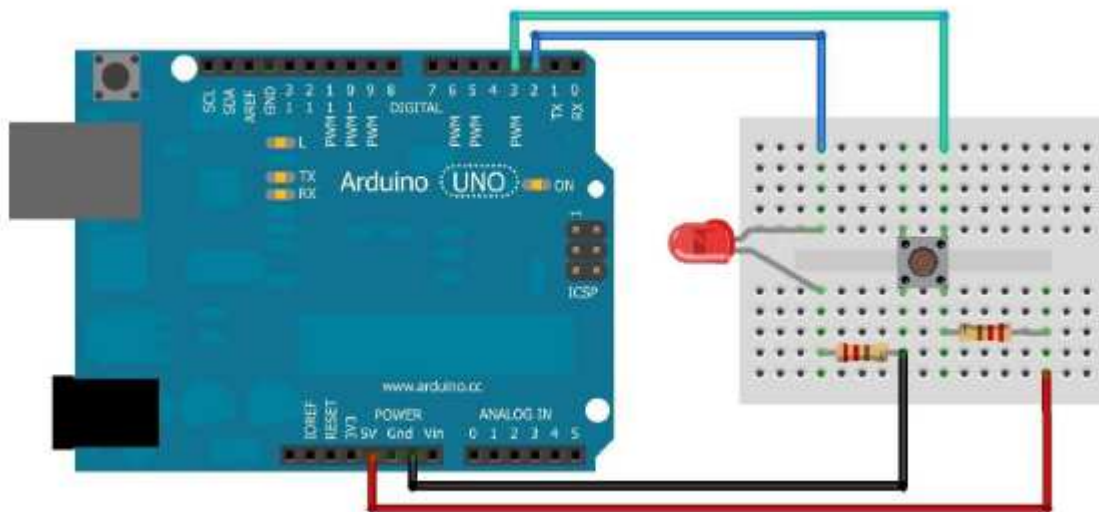
Résistance pull up

Pour des raisons liées aux technologies des circuits logiques, les électroniciens préfèrent utiliser une résistance pull up (résistance de tirage vers le haut) entre l'entrée et l'alimentation (5 V dans le schéma ci-après) et connecter le poussoir comme suit.



La résistance pull up doit aussi avoir une valeur d'au moins 4.7 Kilo-Ohm.

Bouton poussoir et LED



Composants :

Led 1.6V – Résistance 220 Ω

Bouton poussoir 4 broches – Résistance 4.7 k Ω

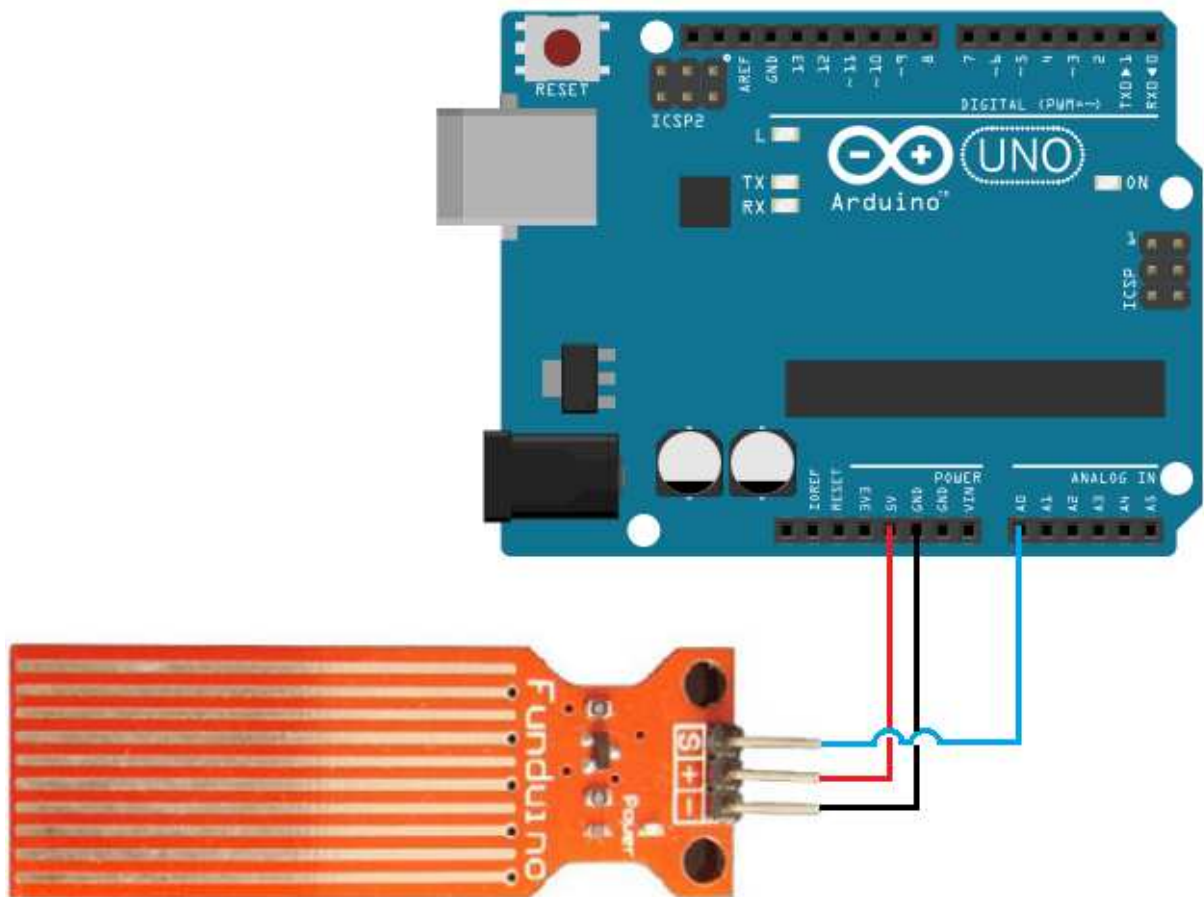
```

sketch_aug15a$
void setup() {
  pinMode(2, OUTPUT); // Configuration de la broche 2 en tant que sortie numerique
  pinMode(3, INPUT_PULLUP); // Configuration de la broche 3 en tant qu'entree numerique.
}
void loop() {
  int buttonState = digitalRead(3); // Lire l'etat de la broche 3.
  if (buttonState == HIGH) { // Changer l'etat de la broche 2 selon l'etat de l'interrupteur
    digitalWrite(2, HIGH); }
  else {
    digitalWrite(2, LOW); }
  }
}

```


TP6 – Le capteur de niveau d'eau

Le but de ce montage est de découvrir comment utiliser le capteur de niveau d'eau. Ce capteur fonctionne de manière analogique. Il envoie une valeur en fonction de la hauteur d'eau.



Principe du montage

Le montage consiste à relier l'alimentation (5V et GND) et une lecture analogique du Arduino au capteur.

- 5V (Arduino) → pin + (Capteur)
- GND (Arduino) → pin – (Capteur)
- Analog 0 (Arduino) → pin S (Capteur)

Programmation

Le programme va lire le capteur et nous renvoyer l'information par la liaison série.

```
// Capteur de niveau d'eau

int capteur =0;// Le capteur est sur la pin A0

int val =0;// Variable de stockage de la valeur lue


void setup ()

{

Serial.begin (9600);// Démarrage de la liaison série

}

void loop ()

{

val = analogRead (capteur);// Lecture de la valeur du capteur

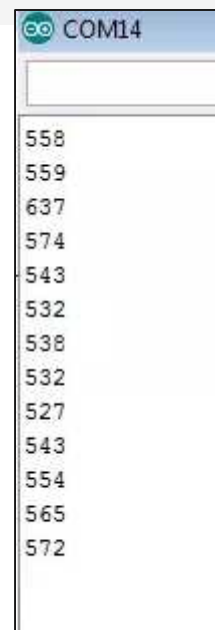
Serial.println (val);// Affichage de la valeur

delay (1000);// Attendre 1s avant la prochaine lecture

}
```

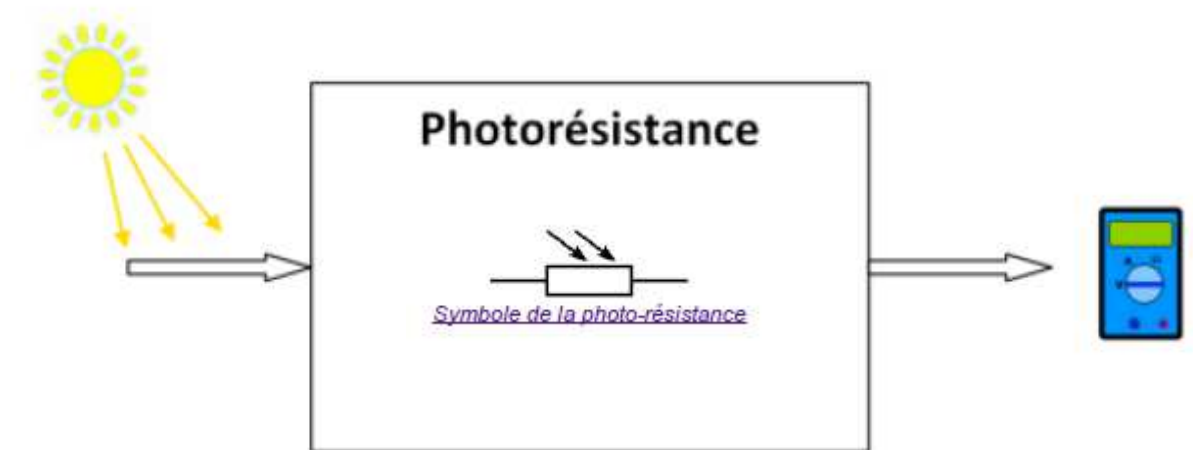
Il reste à brancher le Arduino pour compiler le programme et le téléverser.

Une fois plongé dans l'eau cela nous donne :



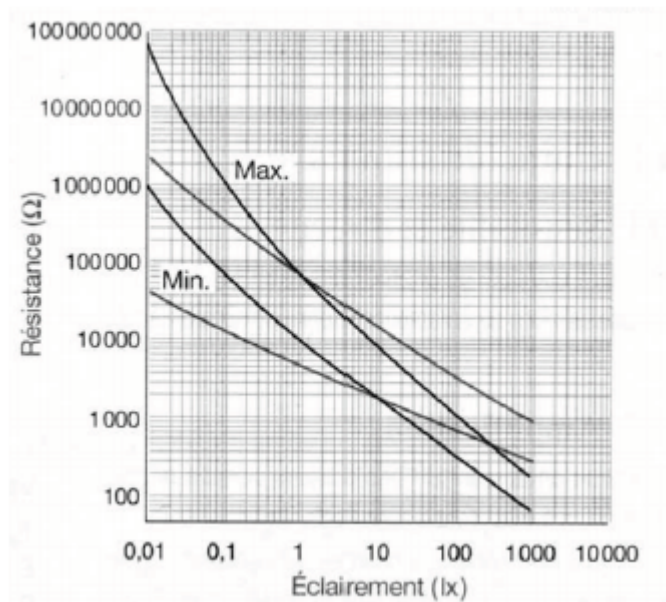
TP7 – La photorésistance

C'est une résistance photosensible, ou si vous préférez qui réagit à la lumière.



On a donc, en sortie du transducteur, une relation du type y en fonction de x : $R = f(E)$. Avec :
 R la résistance en Ohm (Ω)
 E l'intensité lumineuse en lux (lx)

Il s'agit simplement du **rapport** entre la grandeur physique d'entrée du capteur et sa grandeur physique de sortie. Ici, le rapport entre la luminosité et la résistance électrique de sortie. Dans les docs techniques, vous trouverez toujours ce rapport exprimé sous forme graphique (**courbe caractéristique**). Ici, nous avons donc la résistance en fonction de la lumière :



Plus l'intensité lumineuse est élevée, plus la résistance diminue. À l'inverse, plus il fait sombre, plus la résistance augmente.

Malheureusement, les photorésistances ne sont pas des transducteurs très précis. Ils ont notamment des problèmes de linéarité, un temps de réponse qui peut être élevé et une grande tolérance au niveau de la résistance.

Il existe différents types de photorésistances, chacune ayant des valeurs de résistance différentes en fonction de la luminosité ambiante. Le type le plus classique de photorésistances est de 1M ohms (obscurité) / 12K ohms (pleine lumière).

Sans faire une liste exhaustive, voici quelques exemples d'utilisations très classiques pour une photorésistance :

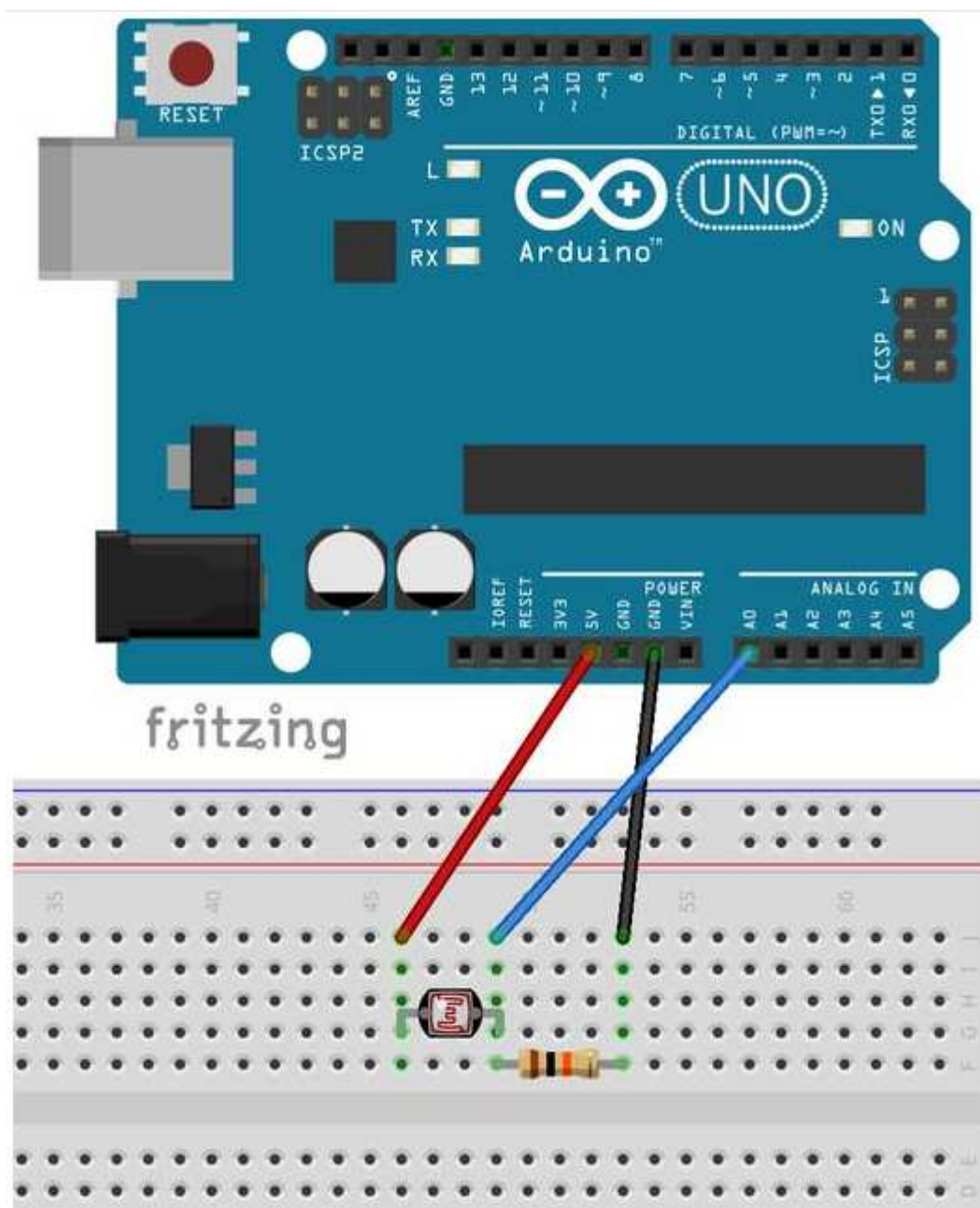
- Détection jour / nuit,
- Mesure de luminosité ambiante (pour ajuster un éclairage par exemple),
- Suiveur de lumière (pour panneaux solaires, robots, etc),

Activité ou lieu concerné	Éclairage moyen
Nuit de pleine lune	0,5 lux
Rue de nuit bien éclairée	20 à 70 lux
Local de vie	100 à 200 lux
Appartement bien éclairé	200 à 400 lux
Local de travail	200 à 3 000 lux
Stade de nuit	150 à 1 500 lux
Extérieur par ciel couvert	500 à 25 000 lux
Extérieur en plein soleil	50 000 à 100 000 lux

Le but du montage de démonstration sera de tout simplement mesurer la luminosité ambiante d'une pièce et d'envoyer la valeur mesurée vers l'ordinateur via le câble USB.

Pour réaliser ce montage, il va nous falloir :

- Une carte Arduino UNO (et son câble USB),
- Une photorésistance de 1M ohms (de diamètre 3mm ou 5mm, cela importe peu),
- Une résistance de 10K ohms (marron / noir / orange),
- Une plaque d'essai et des fils pour câbler notre montage.



electricite.ara@gmail.com

Pour commencer notre montage, nous allons câbler la broche **VCC** de la carte Arduino à une des pattes de la photorésistance au moyen d'un fil.

On relie ensuite la seconde patte de la photorésistance à une des deux pattes de la résistance de 10K ohms. Pour finir, on câble la seconde patte de la résistance de 10K ohms sur la broche **GND** de la carte Arduino.

Une fois la résistance de 10K ohms et la photorésistance câblées, il ne reste plus qu'à relier la jonction entre ces deux résistances à la broche **A0** de la carte Arduino.

Les plus attentifs auront remarqué que ce montage est un pont diviseur de tension.

Le but de notre code va être de :

1. Lire la tension sur la broche **A0**,
2. Envoyer la valeur au PC (pour l'affichage),
3. Recommencer au point 1.

Pour réaliser ce morceau de code, nous allons utiliser la fonction [analogRead\(\)](#) vue précédent.

```
void setup() {  
    Serial.begin(9600);  
}
```

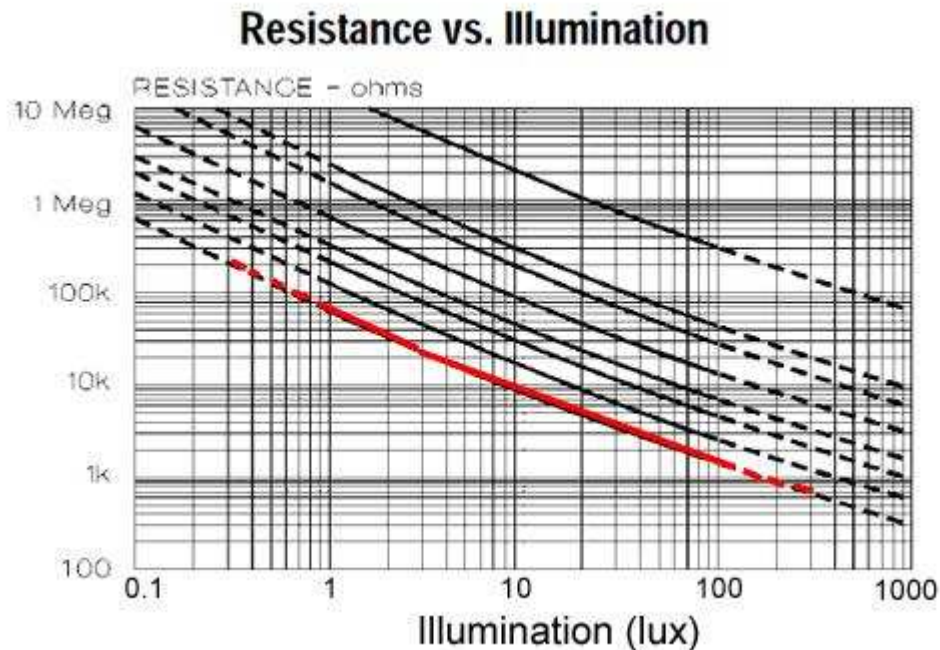
Nous allons commencer notre programme de démonstration avec la fonction **setup()** qui va simplement initialiser la communication avec le PC.

```
void loop() {  
    int valeur = analogRead(A0);  
  
    Serial.println(valeur);  
    delay(250);  
}
```

Dans la fonction `loop()`, nous allons faire deux choses :

1. Mesurer la tension sur la broche **A0** avec **analogRead()**.
2. Envoyer la valeur au PC et attendre quelques millisecondes pour avoir le temps de lire.

Ce code de démonstration est volontairement très simpliste, car il n'y a pas de relation mathématique simple entre la valeur mesurée et la luminosité ambiante.



Si on regarde la courbe de luminosité (en lux) VS la résistance (en ohms) fournie par le fabricant, on se rend compte qu'il ne s'agit pas d'une droite, mais bien d'une courbe. Il est donc assez compliqué de déterminer quelle luminosité (en lux) correspondant à une valeur mesurée par `analogRead()`.

Les photorésistances sont principalement utilisées pour détecter la présence (ou l'absence) de lumière dans une pièce ou à l'extérieur. Pour faire de véritable mesure de luminosité (en lux, avec une précision fixe qu'importent la température et la couleur), il existe des capteurs spécialisés pour cela comme le capteur `TSL2561`

Programmation

```
// Code d'exemple pour une photorésistance.  
// Fonction setup(), appelée au démarrage de la carte Arduino  
void setup() {  
  
    // Initialise la communication avec le PC  
    Serial.begin(9600);  
}  
  
// Fonction loop(), appelée continuellement en boucle tant que la carte  
// Arduino est alimentée  
void loop() {  
  
    // Mesure la tension sur la broche A0  
    int valeur = analogRead(A0);  
  
    // Envoi la mesure au PC pour affichage et attends 250ms  
    Serial.println(valeur);  
    delay(250);  
}
```

Après avoir envoyé le programme dans la carte Arduino, en ouvrant le moniteur série (onglet "outils"), puis en sélectionnant la bonne vitesse de communication (ici 9600 bauds), vous devriez voir apparaître en temps réel la valeur numérique mesurée en sortie de la photorésistance.

Si votre montage est correct, en couvrant la photorésistance ou en la pointant vers une source de lumière, les valeurs dans le moniteur série doivent normalement changer.

N.B. La valeur mesurée n'a pas d'unité ! C'est une valeur purement indicative. Si vous voulez avoir une mesure en lux, il va falloir calibrer votre photorésistance et intégrer les données de calibration dans le code.

Capture d'écran du moniteur série



electricite.ara@gmail.com

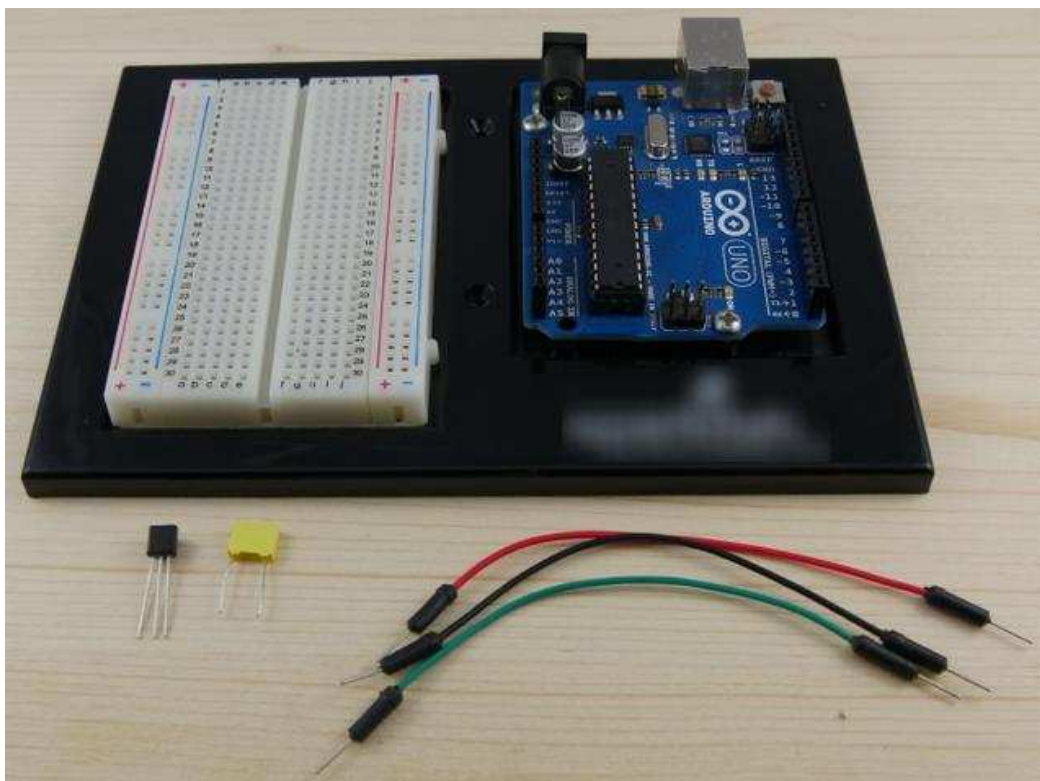
TP8 – La sonde de température LM35

Le capteur de température LM35 est un capteur analogique de température fabriqué par Texas Instruments. Il est extrêmement populaire en électronique, car précis, peu coûteux, très simple d'utilisation et d'une fiabilité à toute épreuve.

Le LM35DZ, capable de mesurer des températures de 0 à 100°C avec une précision de 1.5°C aux extrêmes.

Maintenant que vous savez tout sur le capteur LM35, il est grand temps de le mettre à l'œuvre.

Le but de ce montage sera de tout simplement mesurer la température ambiante de l'atelier et d'envoyer la température en degré Celsius vers l'ordinateur via le câble USB.



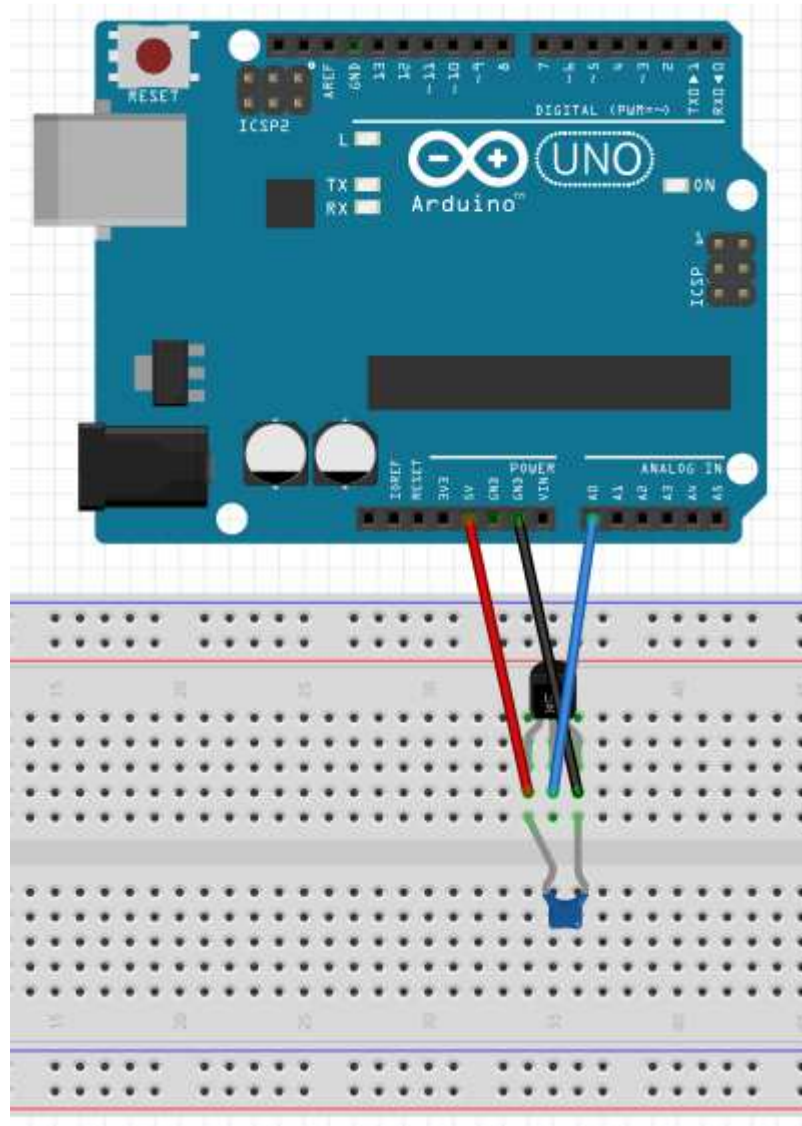
Pour réaliser ce montage, il va nous falloir :

- Une carte Arduino UNO (et son câble USB),
- Un capteur LM35
- Un condensateur de 100nF (optionnel, mais recommandé),
- Une plaque d'essai et des fils pour câbler notre montage.

Pour commencer notre montage, nous allons câbler la broche **VCC** du capteur à l'alimentation 5V de la carte Arduino au moyen d'un fil. On fait ensuite de même avec la broche **GND** du capteur qui vient se câbler sur la broche **GND** de la carte Arduino.

Pour faire les choses bien (parce que oui, on aime faire les choses bien), on va venir câbler un condensateur de 100nF ([un condensateur de découplage](#) en termes techniques) entre les broches **VCC** et **GND** du capteur. Il faut que le condensateur soit câblé le plus près possible du capteur pour être efficace.

On achève ensuite le circuit en reliant la sortie du capteur à la broche **A0** de la carte Arduino avec un fil.



Programmation

Le but de notre code va être de :

1. Lire la tension sur la broche A0
2. Convertir la valeur mesurée en une température (pour l'affichage)
3. Envoyer la valeur au PC (pour l'affichage)
4. Recommencer au point 1.

Dans la fonction setup,

Pour réaliser ce morceau de code, nous allons utiliser la fonction `analogRead()`. Nous allons commencer notre programme de démonstration avec la fonction `setup()` qui va simplement initialiser la communication avec le PC.

```
void setup() {
    Serial.begin(9600);
}
```

Dans la fonction `loop()`, nous allons faire trois choses :

1. Mesurer la tension sur la broche A0 avec `analogRead()`.
2. Transformer le résultat de la mesure en un nombre à virgule (type `float`) en faisant un simple produit en croix. Rappel : $5V = 5000mV = 1023$ en sortie de `analogRead()`, $10mV = 1^{\circ}C$, par conséquent, $température = valeur_mesurée * (5.0 / 1023.0 * 100.0)$
3. Envoyer la valeur au PC et attendre quelques millisecondes pour avoir le temps de lire ce qui se passe côté PC.

```
void loop() {
    int valeur_brute = analogRead(A0);
    float temperature_celcius = valeur_brute * (5.0 / 1023.0 * 100.0);
    Serial.println(temperature_celcius);
    delay(250);
}
```

N.B. On utilise `valeur * (5.0 / 1023.0 * 100.0)` dans le calcul du produit en croix, car lors de la compilation du programme, c'est le type des valeurs d'une opération qui définit le type du résultat. Si on fait `valeur * (5 / 1023 * 100)` comme `valeur`, 5, 1023 et 100 sont des nombres entiers, le résultat est un nombre entier, ce qui n'est pas notre but, nous voulons un calcul avec des nombres à virgule. On utilise donc 5.0, 1023.0 et 100.0 pour forcer un calcul avec des nombres à virgule.

N.B. On multiplie par 100 dans le calcul, car dans 5 volts (= 5000mV) il y a 100 fois 10mV (= 1°C).

Le code complet avec commentaires :

```
// Code d'exemple pour Le capteur LM35 (2°C ~ +110°C).
// Fonction setup(), appelée au démarrage de la carte Arduino
void setup() {
    // Initialise la communication avec le PC
    Serial.begin(9600);
}

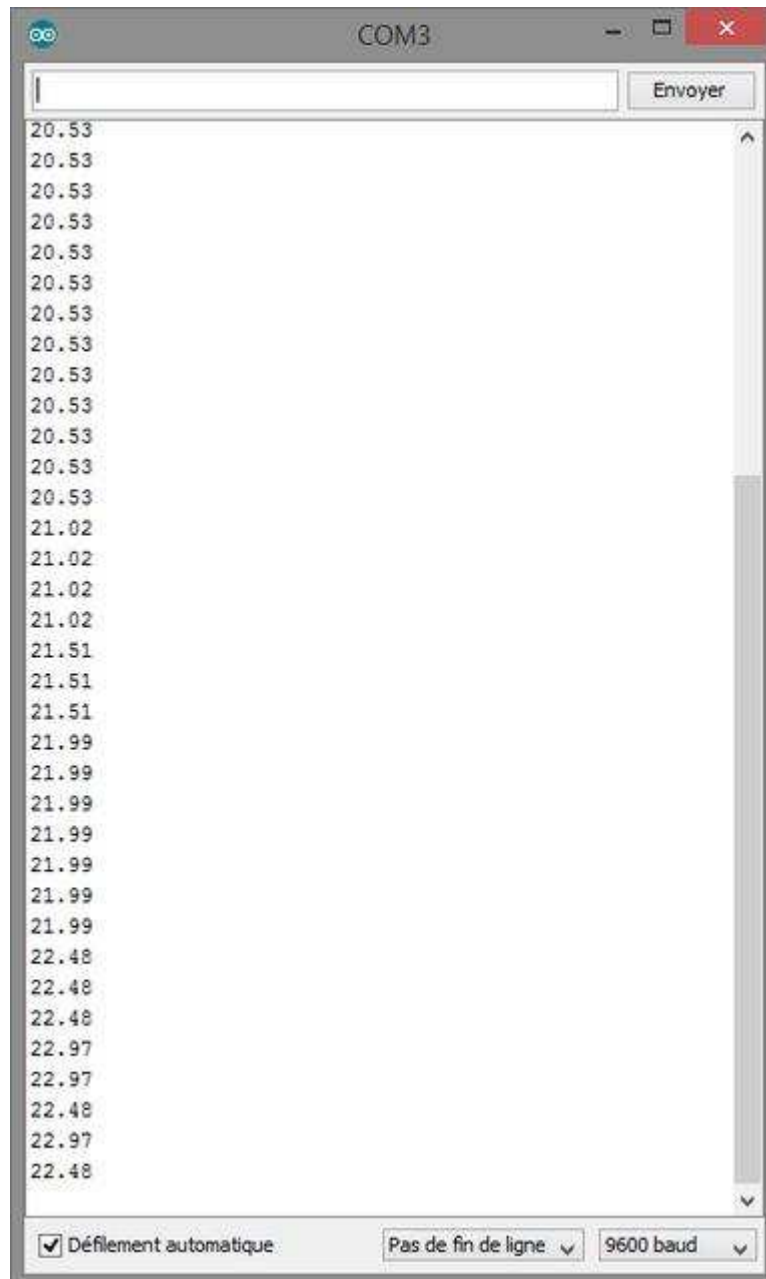
// Fonction loop(), appelée continuellement en boucle
void loop() {

    // Mesure la tension sur la broche A0
    int valeur_brute = analogRead(A0);

    // Transforme la mesure (nombre entier) en température
    float temperature_celcius = valeur_brute * (5.0 / 1023.0 * 100.0);

    // Envoi la mesure au PC pour affichage et attends 250ms
    Serial.println(temperature_celcius);
    delay(250);
}
```

Capture d'écran du moniteur série



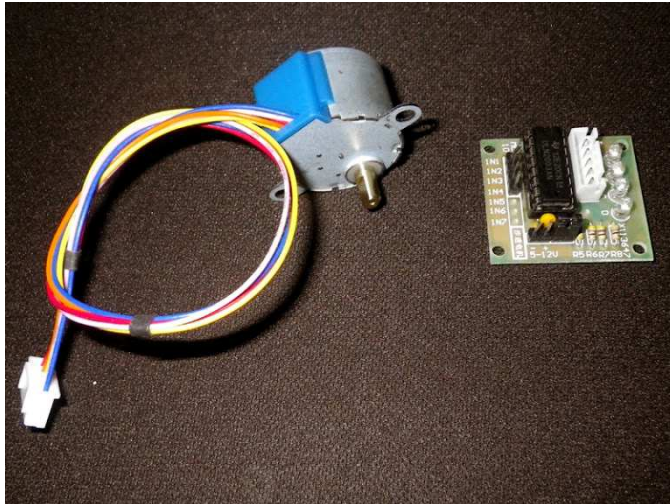
Après avoir envoyé le programme dans la carte Arduino, en ouvrant le moniteur série (onglet **outils**), puis en sélectionnant la bonne vitesse de communication (ici 9600 bauds), vous devriez voir apparaître en temps réel la température en sortie du capteur.

Si votre montage est correct, en pinçant le capteur ou en soufflant dessus, les valeurs dans le moniteur série doivent normalement changer.

PS Evitez de mettre le montage au four à 120°C pour tester les hautes températures, De même, évitez de mettre le montage au congélateur, l'électronique de la carte Arduino n'apprécie pas énormément l'humidité.

TP9 – Le moteur pas à pas

Données sur le moteur 28byj-48 :



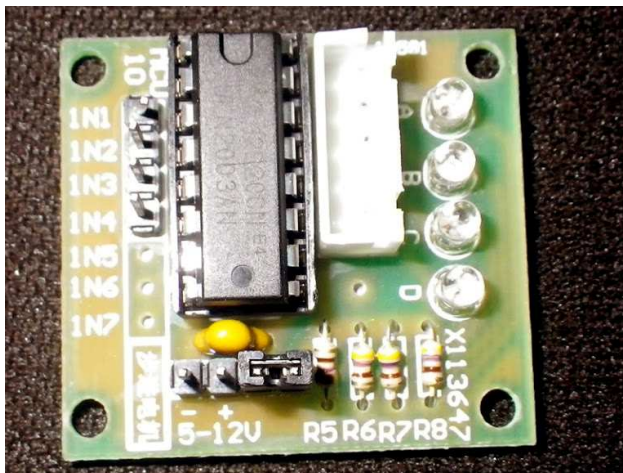
5V DC

4 phases

64 pas par tour

Angle de pas de 5.625°

Données sur le Driver ULN2003 :

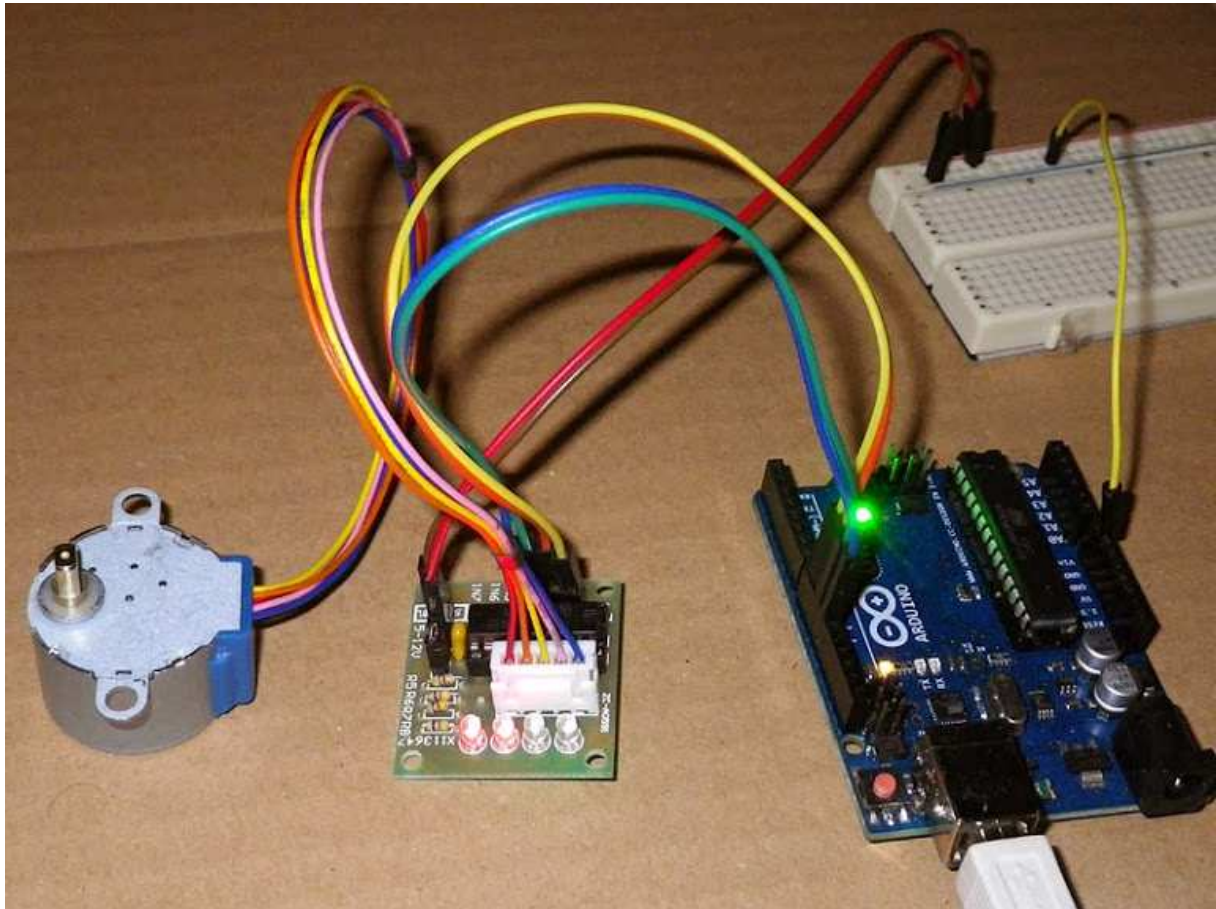


5 broches

MAX 30V 500 mA

Le driver possède des Leds de visualisation et permet de contrôler l'ordre des 4 phases.

4 pins seront reliés à la carte Arduino (IN1, IN2...) et le + et – (5-12V) sur le +5V et le GND de la carte également.

Schéma de câblage :

4 pins seront reliés à la carte Arduino (IN1, IN2...)
Le + et – (5-12V) sur le +5V et le GND de la carte également.

Le programme

```

PasapasTest45degres $
#include <Stepper.h>
#define STEPS 100

Stepper small_stepper(STEPS, 8, 10, 9, 11);    // Sens horaire. Sens anti-horaire en inversant 8 et 11
int Steps2Take = 0; //Nombre de pas de rotation demandé au moteur
long temps =0; //Durée de rotation pour un tour
// 64 pas par tour, 4 phases, angle de 5.625° selon les spécifications du moteur
// Démultiplication 1:64 pour ce moteur réducté mécaniquement
// 360° / 5.625° * 64 = 4096 angles avec la démultiplication
// 360° / 5.625° * 64 * 4 bobines / 2 bipolaire = 2048 step / tour

Première boucle du programme

void setup()
{
  Serial.begin(9600);    // 9600 bps
}

Deuxième boucle du programme

void loop()
{
  small_stepper.setSpeed(150); // Vitesse moteur
  Steps2Take = 7.5*2048/30; // Une rotation complète avec 2048 pas (1 tour environ 4.5sec)
  // Pour tourner à l'envers de 6 fois 1/30eme de tour,
  // simplement multiplier Steps2Take par 6/30 et mettre un moins pour inverser le sens
  // Exemple Steps2Take = -6*2048/30;
  temps = millis();
  small_stepper.step(Steps2Take); //Ca tourne
  delay(2000); //pause de 2 secondes
}

```

Moteur pas à pas : Réglage de la vitesse

Pour ce modèle de moteur :

Le couple décroît avec la vitesse de rotation.

Au-delà d'une vitesse = 300, le moteur ne tourne plus mais vibre : décrochage du pas à pas.

Avec une vitesse = 200, un tour se fait en 6.2 sec.

Avec une vitesse = 100, un tour se fait en 12.4 sec.

Une vitesse très lente = 10 permet de visualiser avec les 4 leds la rotation des phases (effet de chenillard).

TP10 – Le servomoteur

Données sur le servomoteur SG90 :



Dimensions : 22 x 11.5 x 27 mm.

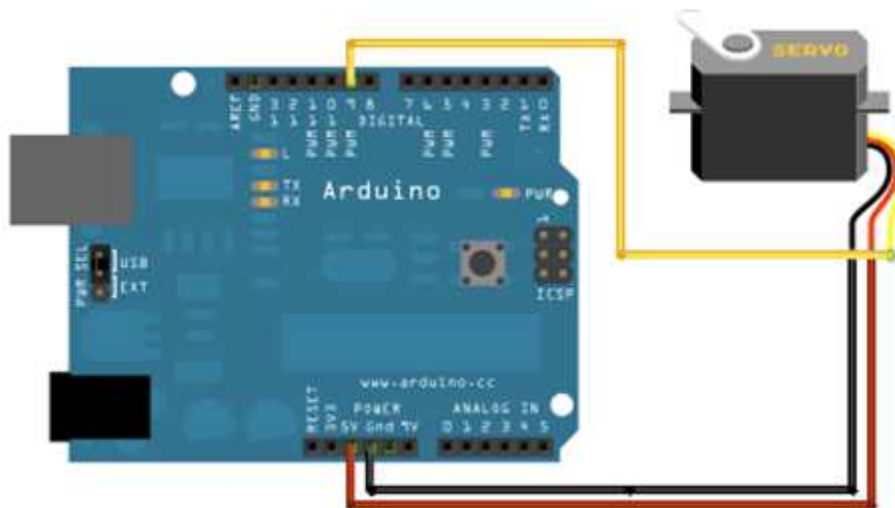
Poids : 9 gr.

Tension d'alimentation : 4.8v à 6v.

Vitesse : 0.12 s / 60° sous 4.8v.

Amplitude : de 0 à 180°.

Schéma de câblage :

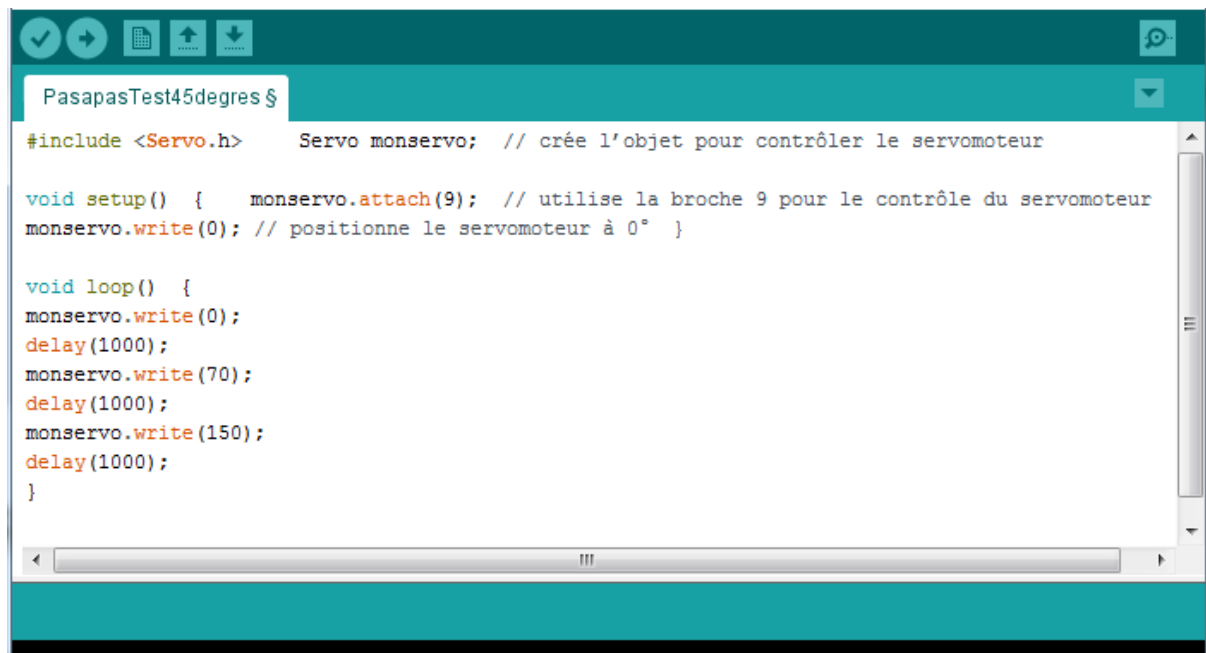


Correspondance des fils :

Brun	Masse GND
Rouge	+ 5V DC
Orange	Fil de commande

electricite.ara@gmail.com

Le programme



```

PasapasTest45degrees $
#include <Servo.h>      Servo monservo; // crée l'objet pour contrôler le servomoteur

void setup() {    monservo.attach(9); // utilise la broche 9 pour le contrôle du servomoteur
monservo.write(0); // positionne le servomoteur à 0° }

void loop() {
monservo.write(0);
delay(1000);
monservo.write(70);
delay(1000);
monservo.write(150);
delay(1000);
}

```

Servomoteur : réglages

- Pour ce modèle de moteur :

Si vous prenez le servomoteur dans votre main, on ressent une vibration car en fait il est arrivé en butée ; cela n'est pas bon pour l'électronique embarqué dans le servomoteur.

Il va donc être nécessaire de découvrir l'angle à adapter pour effectuer véritablement un 180° sans arriver en butée.

Pour ma part, je suis arrivé à une valeur de 150°.

Et pour effectuer un angle de 90°, je dois indiquer une valeur de 70°.

Chapitre 3 : Le robot Timéo

Chapitre 5 : Les exercices

Le multimètre

Le multimètre est l'appareil le plus fréquemment utilisé chez l'électricien. Il s'agit d'un appareil comprenant plusieurs autres appareils de mesure au sein d'un même boîtier. Pour

Chapitre 6 : Compétences à atteindre en fin de cycle

Résistance de terre

L'araignée Gipsy

Le multimètre est l'appareil le plus fréquemment utilisé chez l'électricien. Il s'agit d'un appareil comprenant plusieurs autres appareils de mesure au sein d'un même boîtier. Pour