Athénée Royal d'Ans

« Electronique »

« ARDUINO »

« Théorie & Laboratoire »

6ème année technique

7ème année professionnelle

Année scolaire :

Classe : 6 TEA - 7 PEL

Professeur : C. Baldewijns - F. Caprace - R. Radoux

1. Sommaire

2. Objectifs 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 les bases de programmation en langage C/C++ à des élèves de 1ère et 2ème année en énonçant les bases simplifiées de celle-ci.

Ce cours n'est qu'un aperçu de ce qu'on peut faire avec la carte Arduino. Pour plus d'informations, n'hésitez pas à parcourir les documents mis à disposition sur le serveur de l'école. Votre professeur vous indiquera où trouver ces documents.

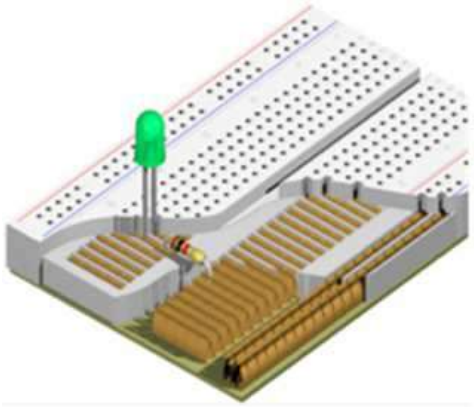
- ***Sources - Bibliographies :***

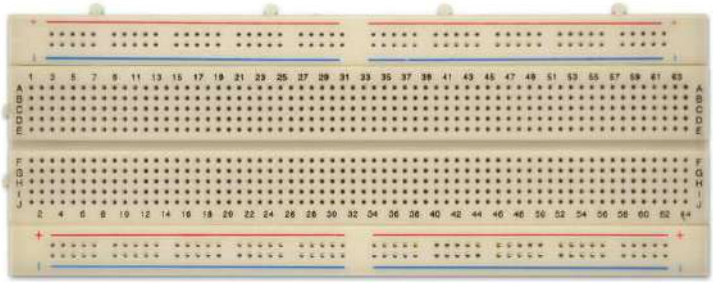
- Le grand livre d'Arduino (2ème édition) (Eyrolles), 2015, 613 pages. (BARTMANN Erik)

- Arduino, premiers pas en informatique embarquée (blog d'Eskimon), 19 juin 2014, 454 pages. (LANDRAULT Simon et WEISSLINGER Hippolyte)

3. Matériels et composants

☞ ***La breadboard*** (platine de montage)





Par convention :

Noir = masse
Rouge = alimentation (+5V, +12V, 5V… ce que vous voulez y amener).

Remarque : Un espace coupe la carte en deux de manière symétrique. Cette espace coupe 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 liaison***

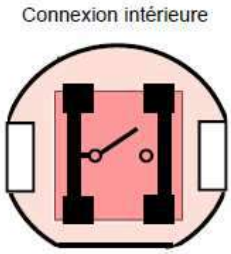
Pour faire la jonction entre les composants électroniques et la breadboard, nous utiliserons des fils de couleurs assez fins

☞ *Le bouton poussoir*

Un bouton poussoir laisser passer le courant lorsqu'il est poussé et l'interrompt quand il est relâché

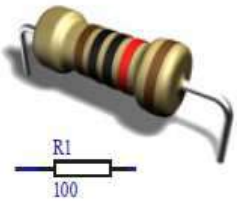
On peut aussi trouver des boutons poussoirs NO ou NF, ces boutons poussoirs vont nous servir à faire passez des courants électriques dans nos composants électroniques grâce à une action manuelle sur l'extrémité de ceux-ci.

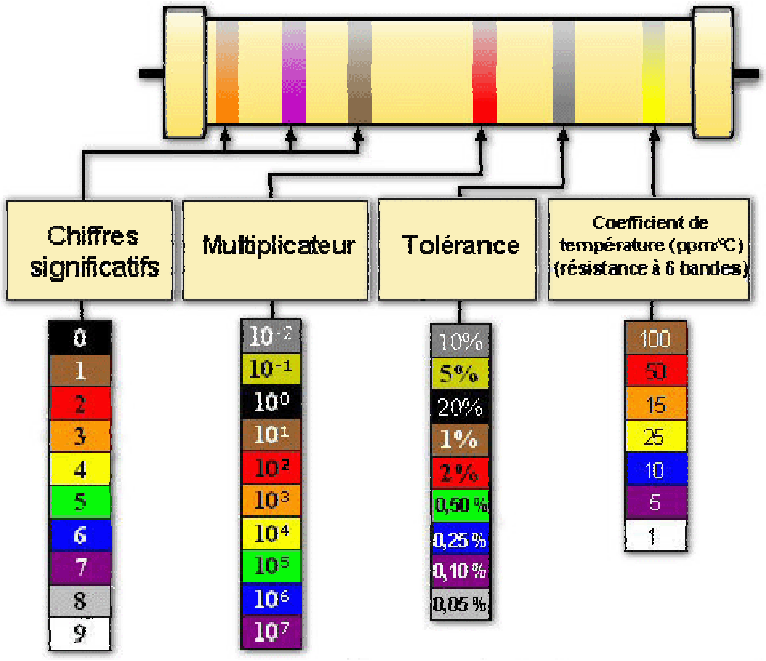
Le bouton fourni a 4 connecteurs, mais seulement 2 sont utilisés. 2 sont interconnectés



☞ *La résistance (R)*

La résistance s'oppose au passage du courant, proportionnellement à sa " résistance" exprimée en Ohm. Un code de couleurs, ci-dessous, permet de reconnaître cette valeur. La résistance est un composant non polarisée

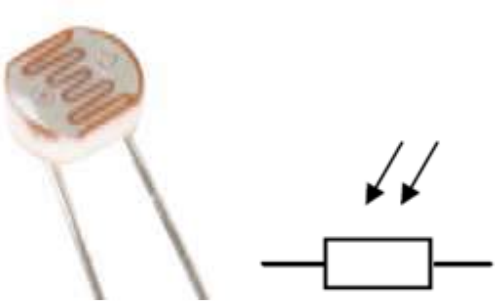




☞ *La LDR (photorésistance)*

C'est une résistance variable, en fonction de la luminosité qu'elle reçoit. Sa résistance diminue quand elle reçoit de la lumière.

On s'en sert donc de capteur de luminosité. Non polarisée. Pour lire sa valeur avec une Arduino, il faut également l'associer avec une résistance équivalente à sa résistance maximum (dans le noir)



☞ *Le potentiomètre*

Le potentiomètre, rotatif ou à glissière, est une résistance variable.

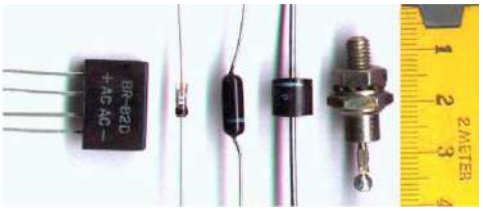
Entre les extrémités, il y a la résistance maximale. La patte centrale est le curseur. C'est la résistance entre cette patte centrale et une extrémité que l'on peut faire varier en tournant le bouton.

Le potentiomètre est donc un capteur. Il se branche sur les entrées analogiques de l'Arduino.

De très nombreux capteurs sont basés sur le principe de résistance variable et se câblent presque de la même façon: la cellule photo-électrique, le capteur de pression, le fil résistif, etc

☞ *La diode*

La diode ne laisse passer le courant que dans un seul sens. C'est un composant polarisé: on reconnait toujours son anneau coloré d'un coté du composant, correspondant à la cathode.



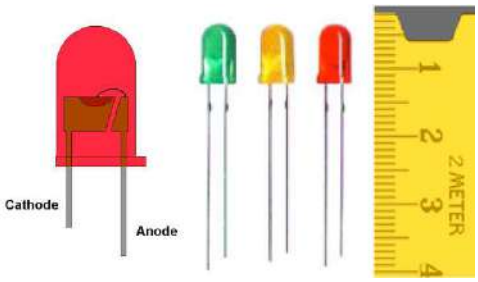
☞ ***La diode LED***

La diode électroluminescente (LED) émet de la lumière. Elle est polarisée:

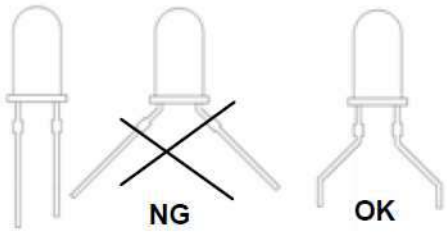
La patte “+” est la plus longue, l’autre patte est la patte “-” .

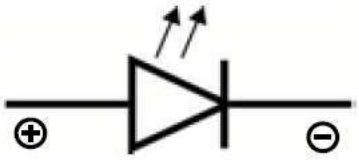
Les broches numériques de l’Arduino, lorsqu’elles sont configurées en sorties et qu’elles sont à l’état 1 ou haut (HIGH) , fournissent une tension de 5 volts, supérieure à ce que peut accepter une LED.

Les LED doivent donc être couplées en série avec une résistance.



Comment plier les tiges :



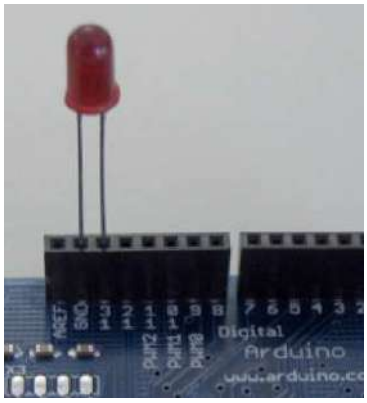


La tension de seuil dépend de la couleur et donc de la composition chimique du dopage .

Couleurs	***Tension de seuil ou Vf***	***If (mA)***	***Longueur d'onde (nm)***
Rouge	1,6 à 2V	6 à 20 mA	650 à 660 nm
Jaune	1,8 à 2V	6 à 20 mA	565 à 570 nm
Vert	1,8 à 2V	6 à 20 mA	585 à 590 nm
Bleu	2,7 à 3,2V	6 à 20 mA	470 nm
Blanc	3,5 à 3,8V	6 à 20 mA	

Remarque : broche 13 de l'Arduino

La broche numérique 13 de l’Arduino est déjà câblée en série avec une résistance de valeur moyenne pour une LED (1 kΩ), on peut donc, dans la plupart des cas, directement lui brancher une LED, comme sur la photo-ci-dessous. Attention de respecter la polarité !!

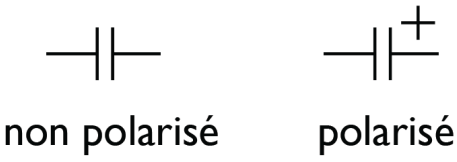


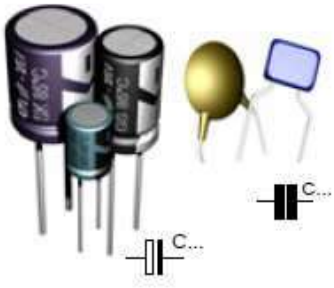
Il ne reste plus qu’à déclarer dans le programme que la broche 13 est configurée en sortie, et le tour est joué pour faire quelques essais.

Sur les autres broches, il faut dimensionner la résistance en série avec la diode !!

☞ *Le condensateur (C)*

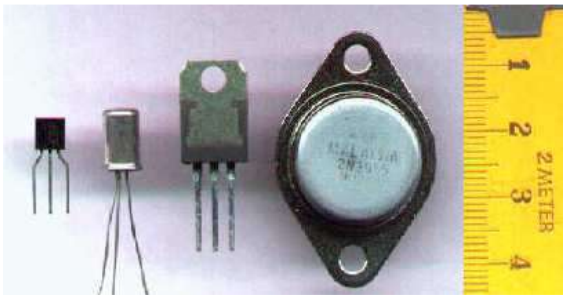
Les condensateurs peuvent stocker un peu de courant si on les charge, mais comme un tonneau percé, ils renvoient ce courant instantanément si ils sont branchés à un organe consommateur de courant.
Les condensateurs sont aussi utilisés pour stabiliser ou filtrer des tensions non désirées. L'unité de sa capacité est le farad, mais les valeurs usuelles sont le µF, nF, pF. Le condensateur peut être polarisé ou non en fonction de sa conception.

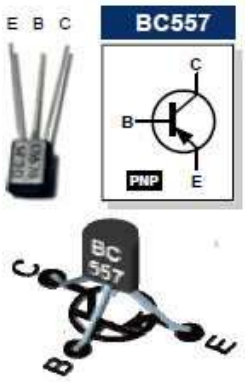


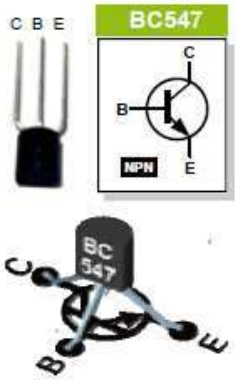


☞ *Le transistor*

Le transistor sert à amplifier un signal. Un faible courant de commande peut ainsi être transformé en un courant plus important. On distingue 2 types de transistors, selon leur polarité. Le NPN et le PNP. Un transistor possède 3 pattes : la base (B), l'émetteur (E) et le collecteur (C)







☞ *Le relais*

Le relais est un composant à 4 broches minimum. C'est un électroaimant que l'on peut commander en envoyant un petit courant.

Au repos, il est normalement fermé, ou normalement ouvert, selon le modèle.

On peut s'en servir avec l'Arduino pour commander des machines en haute tension (230V par exemple), ou pour déclencher toute machine ou lumière.

☞ *Le servomoteur*

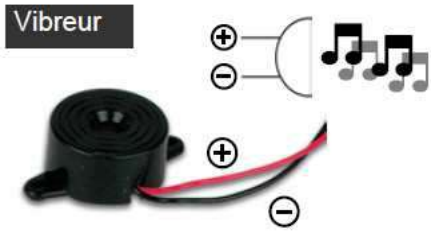
Le servo-moteur est un moteur (rotatif) qui peut effectuer des rotations très précises (dans une portion de tour seulement) et en un certain nombre de pas (de micro-déplacements).

Il y a toutes sortes de servo moteurs.. Un des avantages des servo moteurs est sa possibilité de maintenir avec force une position donnée.

On peut piloter des rotations avec l'Arduino, quelques fois directement avec la carte si le moteur n'est pas trop puissant, sinon en passant par un montage associé. (shields)

☞ *Le vibreur*

Un vibreur produit un signal sonore pour alerter d'une situation dangereuse, pour confirmer une minuterie, quand un bouton est pressé, ... La tonalité du vibreur ne peut être modifiée, car la fréquence de l'oscillateur est fixée.



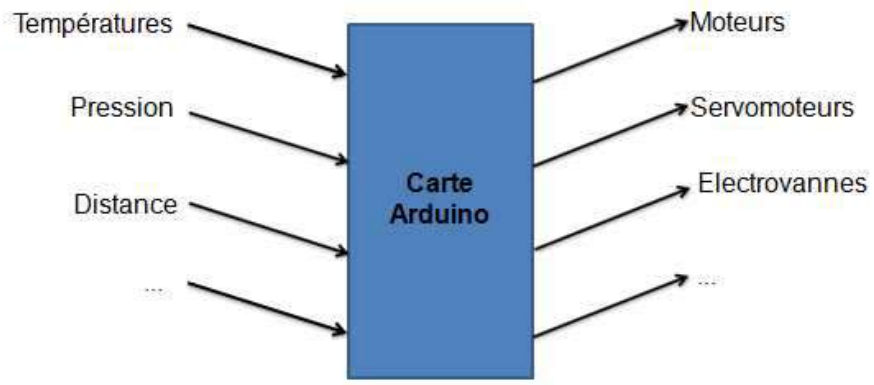
☞ *Le piezo*

Le transducteur piezo-électrique est un composant réversible: il peut aussi bien être utilisé en capteur de chocs ou de vibrations qu'en actionneur pouvant émettre des sons stridents parfois modulables.

4. ARDUINO

4.1. Ca sert à quoi ?

Interagir avec le monde réel



4.2. C'est pour qui ?

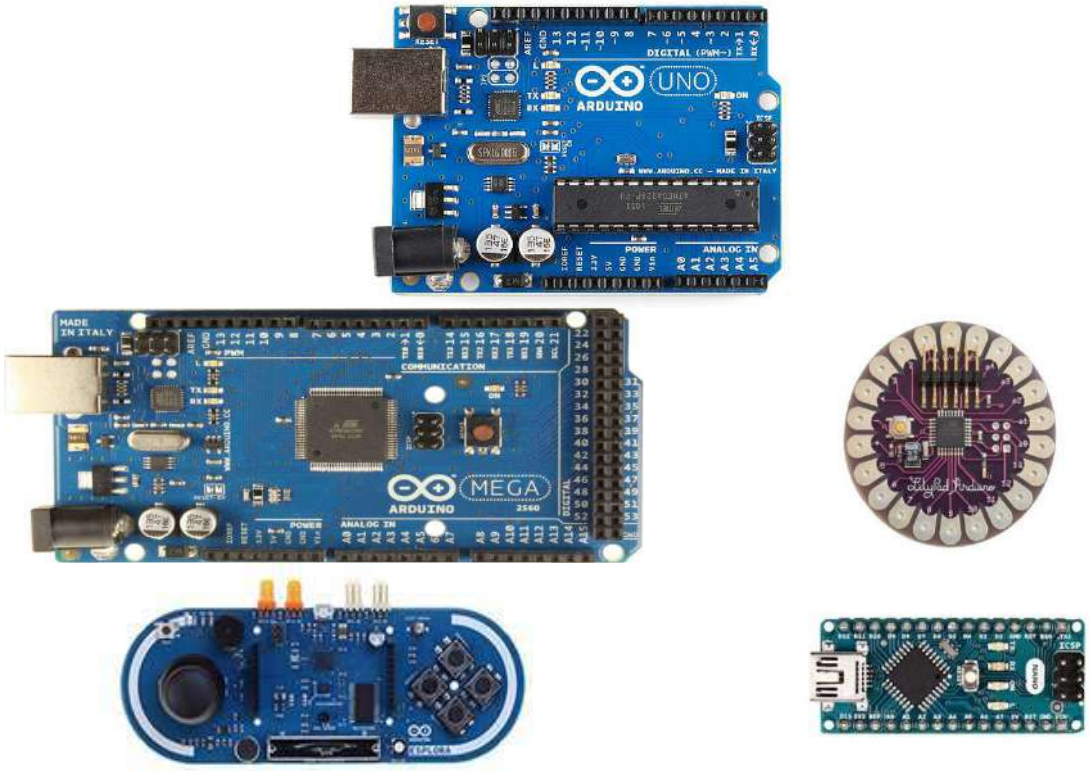
Pour les jeunes et les adultes

Tout le monde peut faire ses premières armes avec les composants électroniques et s'initier à leur programmation.

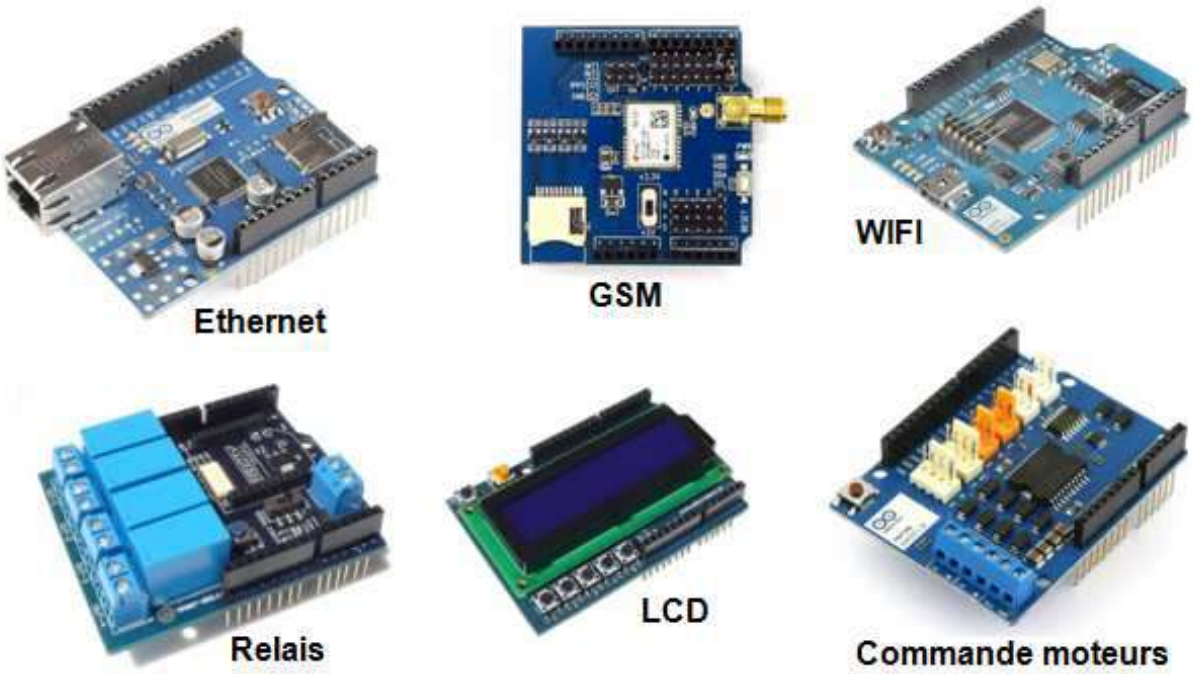
4.3. Qu'est-ce que c'est

- ☞ Une plate-forme de développement et de prototypage Open Source.
- ☞ Le rôle de la carte Arduino est de stocker un programme et de le faire fonctionner.
- ☞ Shields (cartes d'extension) avec des fonctions diverses qui s'enfichent sur la carte Arduino :
  - Relais, commande de moteurs, lecteur carte SD, ...
  - Ethernet, WIFI, GSM, GPS, …
  - Afficheurs LCD, Écran TFT, ...

4.3.1. Quelques cartes Arduino



4.3.2. Divers Shields Arduino



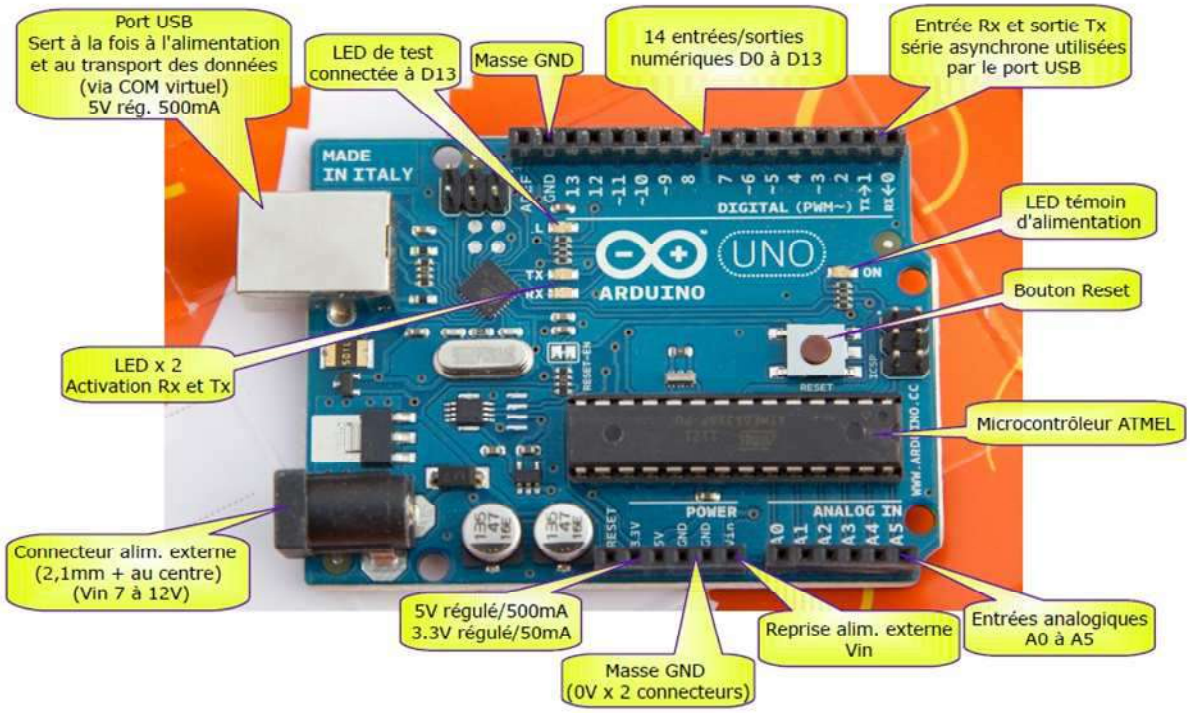
4.4. Présentation de la carte UNO

L'Arduino est une plateforme open source d'électronique programmée qui est basée sur une carte à microcontrôleur et un logiciel. Plus simplement, on peut dire que l'Arduino est un module électronique, doté d'un microcontrôleur programmable.

Il peut être utilisé pour développer des objets interactifs, munis d'interrupteurs ou de capteurs, et peut contrôler une grande variété de lumières, moteurs ou toutes autres sorties matérielles.

La programmation se fait à l'aide d'un langage proche du C/C++, dont les bases sont faciles d'accès. Le logiciel nécessaire fonctionne à la fois sur Mac OSX, Windows et GNU/Linux et demande très peu de ressources.

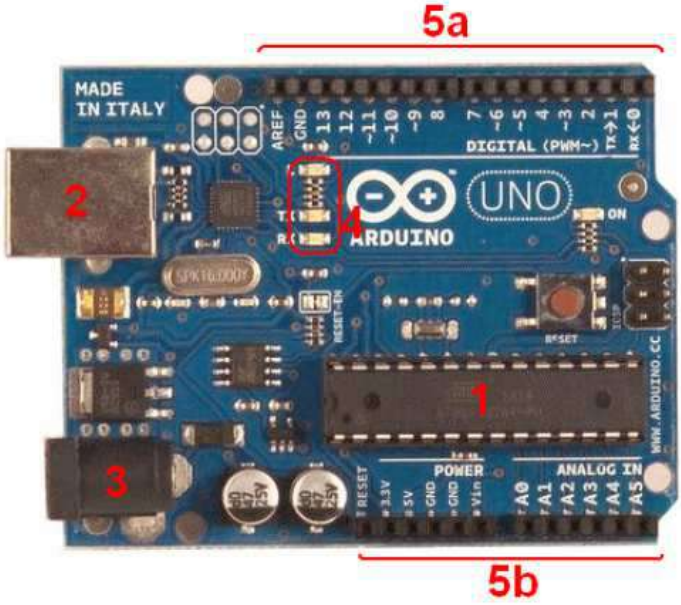
Comme le prix d'un module Arduino est faible , nous disposons donc d'un excellent outil qui nous permettra d'apprendre les bases de la programmation, de l'électricité, de l'électronique et de l'automation.



Le principe de fonctionnement est simple:

1. On réalise le programme sur un ordinateur.
2. On connecte l'ordinateur à l'Arduino via une prise USB.
3. On envoie le programme sur l'Arduino.
4. L'Arduino exécute enfin le programme de manière autonome.

4.4.1. Quelques précisions sur 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 a par 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.

❖ ***Les entrées / les sorties digitales***

Ce sont les deux rangées de connecteurs de part et d'autre de la carte qui permettent sa connexion au monde extérieur. (D0 à D13)

Chacun des connecteurs D0 à D13 peut être configuré dynamiquement par programmation en entrée ou en sortie.

Les signaux véhiculés par ces connecteurs sont des signaux logiques compatibles TTL, c'est-à-dire qu'ils ne peuvent prendre que deux états HAUT (5 Volts) ou BAS (0 Volt).

En pratique, les connecteurs D0 et D1 réservés pour la liaison série asynchrone (port COM virtuel via le câble USB) ne sont pas exploités pour d'autres utilisations.

Certains connecteurs peuvent être spécialisés comme sorties PWM (repérées par un ~) .

Attention : chacun des connecteurs ne peut fournir ou absorber un courant supérieur à **40 mA**

❖ ***Les entrées analogiques***

Par défaut et contrairement aux entrées/sorties digitales qui ne peuvent prendre que deux états HAUT et BAS, ces six entrées peuvent admettre toute tension analogique comprise entre 0 et 5 Volts.

❖ ***Les bornes d'alimentation***

- GND : Terre (0V)
- 5V : Alimentation +5V
- 3.3V : Alimentation +3.3V
- Vin : Alimentation non stabilisée (= même tension que celle à l'entrée de la carte)

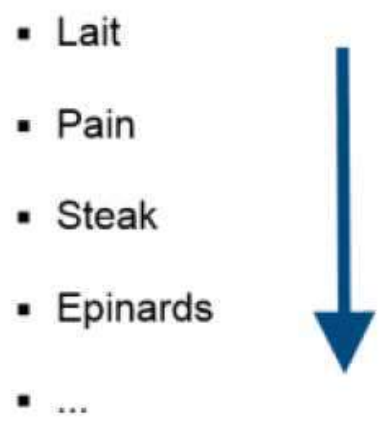
4.5. La programmation, qu'est-ce que c'est ?

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 enchainer 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 :



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 écris 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 icone :

Cliquez deux fois dessus pour accéder aux lignes de code.

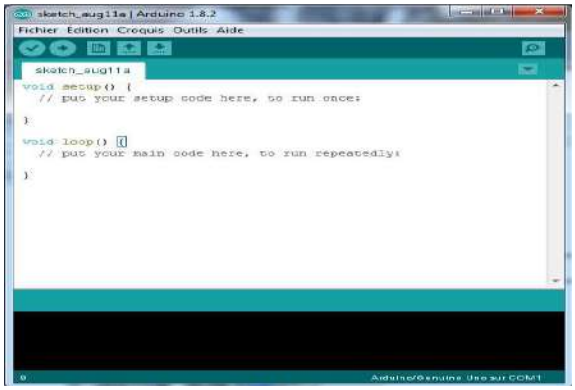
4.6. Le langage C

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 !

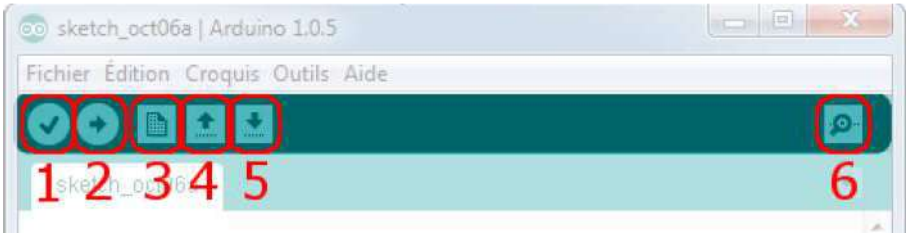
Dans ce cours, vous commencerez par découvrir le fonctionnement de la mémoire, des variables, des conditions et des boucles.

4.7. L'environnement de programmation

Programmer avec le logiciel ***IDE Arduino***



L'IDE (environnement de développement intégré) fonctionnant sur divers systèmes d'exploitation qui permet d'éditer le programme sur un ordinateur et de le transférer via le port usb.



1		***Verify :*** Ce bouton permet de vérifier le programme, il actionne un module qui cherche les erreurs dans votre programme
2		***Upload :*** Charge (téléverse) le programme dans la carte Arduino
3		***New :*** pour créer un nouveau programme (sketch)
4		***Open :*** pour ouvrir un programme existant
5		***Save :*** pour sauvegarder le programme
6		***Verify :*** pour visualiser les données transmises par le port série

<u>Coloration syntaxique de l'IDE:</u>

En **orange**, apparaissent les mots-clés reconnus par le langage Arduino comme les **fonctions** .

En **bleu**, apparaissent les mots-clés reconnus par le langage Arduino comme des **constantes**.

En **gris**, les **commentaires** qui ne seront **pas exécutés dans le programme**.
Déclarer un commentaire de deux manières différentes :

- tout ce qui se trouve après « ***//*** » sera un commentaire.
- on encadrer des commentaires sur plusieurs lignes entre « ***/**** » et « ****/*** ».

5. Instructions de base

Sur le serveur de l'école se trouvent deux documents pdf reprenant l'ensembles des instructions permises avec Arduino. Il est impératif de les consulter quand le besoin s'en fait sentir !! (***Instructions de base - Arduino*** et ***Arduino - Références (Liste des instructions)***)

5.1. Liste d'instructions

Fonctions de base Ces deux fonctions sont obligatoires dans tout programme en langage Arduino : • void setup() • void loop()	**Structures de contrôle** • if • if...else • for • switch case • while • do... while • break • continue • return • goto	**Syntaxe de base** • ; (point virgule) • {} (accolades) • // (commentaire sur une ligne) • /* */ (commentaire sur plusieurs lignes) • #define • #include
Opérateurs arithmétiques • = (égalité) • + (addition) • - (soustraction) • * (multiplication) • / (division) • % (modulo)	**Opérateurs de comparaison** • == (égal à) • != (différent de) • < (inférieur à) • > (supérieur à) • <= (inférieur ou égal à) • >= (supérieur ou égal à)	**Opérateurs booléens** • && (ET booléen) • || (OU booléen) • ! (NON booléen)
Pointeurs • * pointeur • & pointeur Voir également : • Manipulation des Ports	**Opérateurs bit à bit** • & (ET bit à bit) • |(OU bit à bit) • ^(OU EXCLUSIF bit àbit) • ~(NON bit à bit) • << (décalage à gauche) • >> (décalage à droite)	**Opérateurs composés** • ++ (incrémentation) • -- (décrémentation) (à revoir) • += (addition composée) • -= (soustraction composée) • *= (multiplication composée) • /= (division composée) • &= (ET bit à bitcomposé) • |= (OU bit à bit composé)

Variables et constantes

Les variables sont des expressions que vous pouvez utilisez dans les programmes pour stocker des valeurs, telles que la tension de sortie d'un capteur présente sur une broche analogique.

Constantes prédéfinies

Les constantes prédéfinies du langage Arduino sont des valeurs particulières ayant une signification spécifique.

- HIGH | LOW
- INPUT | OUTPUT
- true | false

A ajouter : constantes décimales prédéfinies

Expressions numériques

- Expressions numériques entières
- Expressions numériques à virgule

Types des données

Les variables peuvent être de type variés qui sont décrits ci-dessous.

Synthèse des types de données Arduino

- boolean
- char
- byte
- int
- unsigned int
- long
- unsigned long
- float (nombres à virgules)
- double (nombres à virgules)
- Les chaînes de caractères
- objet String NEW
- Les tableaux de variables
- le mot-clé void (fonctions)
- word
- PROGMEM

Voir également :

- Déclaration des variables

Pour info : les types de données avr-c

Conversion des types de données

- char()
- byte()
- int()
- long()
- float()
- word()

Portée des variables et qualificateurs

- Portée des variables
- static
- volatile
- const

Utilitaires

- sizeof() (opérateur sizeof)

Référence

- Code ASCII

Fonctions

Entrées/Sorties Numériques

- pinMode(broche, mode)
- digitalWrite(broche, valeur)
- int digitalRead(broche)

Entrées analogiques

- int analogRead(broche)
- analogReference(type)

Sorties "analogiques" (génération d'impulsion)

- analogWrite(broche, valeur) - PWM

Entrées/Sorties Avancées

- tone()
- noTone()
- shiftOut(broche, BrocheHorloge, OrdreBit, valeur)
- unsigned long pulseIn(broche, valeur)

Communication

- Serial

Temps

- unsigned long millis()
- unsigned long micros()
- delay(ms)
- delayMicroseconds(us)

Math

- min(x, y)
- max(x, y)
- abs(x)
- constrain(x, a, b)
- map(valeur, toLow, fromHigh, toLow, toHigh)
- pow(base, exposant)
- sq(x)
- sqrt(x)

Pour davantage de fonctions mathématiques, voir aussi la librairie math.h : log, log10, asin, atan, acos, etc...

Nombres randomisés (hasard)

- randomSeed(seed)
- long random(max)
- long random(min, max)

Trigonométrie

- sin(rad)
- cos(rad)
- tan(rad)

Bits et Octets

- lowByte()
- highByte()
- bitRead()
- bitWrite()
- bitSet()
- bitClear()
- bit()

Interruptions Externes

- attachInterrupt(interruption, fonction, mode)
- detachInterrupt(interruption)

Interruptions

- interrupts()
- noInterrupts()

Voir également la librairie interrupt.h.

5.2. Résumé des instructions

Voici cependant un résumé très rapide des principales instructions que nous utiliserons

☞ ***Syntaxe du langage (Arduino)***

La syntaxe d'un langage de programmation est l'ensemble des règles d'écritures liées à ce langage. On va donc voir dans ce sous chapitre les règles qui régissent l'écriture du langage Arduino.
La structure de base du langage de programmation Arduino est assez simple et comprend au moins deux parties. Ces deux parties, ou fonctions, contiennent des blocs d'instructions.

Setup() est l'initialisation du programme et loop() est l'exécution du programme. Ces deux fonctions sont impérativement requises pour que le programme fonctionne.

La fonction setup() doit suivre la déclaration des variable au tout début du programme. Il s'agit de la première fonction à exécuter dans le programme. Elle est exécutée une seule fois et sert à établir le mode d'une broche (pinMode) ou à initialiser la communication série.

La fonction loop() suit immédiatement et comprend le code à exécuter en continue - lisant les capteurs en entrée et déclenchant les actionneurs en sortie, etc. Cette fonction est le noyau de tout programme Arduino et réalise l'essentiel du travail.

```
void setup()
{
blocs d'instructions;
}
void loop()
{
blocs d'instructions;
}
```

- **Setup**

La fonction setup() n'est appelée qu'une fois au démarrage du programme. Utilisez-là pour initialiser le mode des broches ou fixer le débit de communication série. Il doit être dans tout les programmes même si il n'y a aucune instruction à jouer.

```
void setup()
{
pinMode(broche, OUTPUT); //met la "broche" comme sortie
}
```

- **Loop**

Après avoir appelé la fonction void setup(), la fonction void loop() fait une boucle (loop en anglais) infinie permettant de contrôler la carte arduino, tant que le programme ne change pas.

Les fonctions void setup() et void loop() sont les 2 fonctions obligatoires en arduino, mais d'autres peuvent être créées.

```
void loop()
{
digitalWrite(broche, HIGH); //met la broche en "ON"
delay(1000); //pause pendant une seconde
digitalWrite(broche, LOW); //met la broche en "OFF"
delay(1000); //pause pendant une seconde
}
```

☞ Les accolades {}

Les accolades (ouvrantes ou fermantes) définissent le début et la fin d'une fonction ou d'un bloc d'instructions, comme pour la fonction void loop() ou les instructions if et for.

Une accolade ouvrante { est toujours suivie par une accolade fermante }. Une accolade non fermée donne des erreurs ou des programmes lors de la compilation mais peut être dure à corriger en cas de programmes très longs.

L'environnement Arduino inclue un système de vérification des accolades. Il suffit de cliquer sur une accolade pour voir sa "paire" se surligner.

```
type fonction()
{
instructions;
}
```

☞ Le point-virgule

Le point-virgule est utilisé pour finir une instruction et séparer les éléments d'un programme. On l'utilise, par exemple, pour séparer les éléments de la boucle for.

```
int x = 13; //on déclare la variable x comme un entier valant 13
```

Note : Oublier le point-virgule à la fin d'une ligne donne une erreur de compilation. Le message d'erreur peut parfois faire référence au point-virgule oublié, mais pas toujours. Si le message d'erreur est illogique et incompréhensible, une des premières choses à faire est de vérifier s'il n'y a pas un point-virgule manquant.

☞ Les paragraphes commentaires /*.....*/

Un commentaire (sur une ligne ou sur plusieurs) est une zone de texte qui est ignoré par le programme lors de la compilation et qui est utilisé afin de décrire le code ou pour le commenter pour faciliter sa compréhension auprès de personnes tiers. Un commentaire commence par /* et fini avec */ et peut faire plusieurs ligne.

```
/* ceci est un paragraphe de commentaire commencé
ne pas oublier de le refermer
*/
```

De par le fait que les commentaires soient ignorés et qu'ils ne prennent pas d'espace mémoire, ils peuvent être utilisés généreusement, par exemple pour débuger une partie du code.
Note : Il est possible de faire un paragraphe d'une seule ligne avec /* */ mais on ne peut pas inclure un second bloc dans le premier.

☞ **Les lignes de commentaires // .....**

Une seule ligne de commentaire commence par // et fini à la fin de la ligne de code. Tout comme le paragraphe de commentaires, la ligne est ignorée lors de la compilation et ne prend pas de mémoire.

La ligne de commentaire est souvent utilisée pour donner des informations ou pour rappeler quelque chose à propos d'une ligne.

```
//ceci est une ligne de commentaires
```

☞ **Les variables**

Une variable est une façon de nommer et de stocker une information/valeur afin de l'utiliser plus tard dans le programme. Comme son nom l'indique, une variable est une valeur qui peut être continuellement modifiée à l'opposé d'une constante qui ne change jamais de valeur. Une variable a besoin d'être déclarée et on peut lui assigner une valeur que l'on veut stocker, mais cela est optionnel. Le code suivant déclare une variable nommé inputVariable et lui assigne une valeur lue via analogRead sur la broche 2.

```
int inputVariable = 0; //déclare la variable et lui assigne la valeur 0
inputVariable = analogRead(2); //assigne la valeur de la broche analogique 2 à la variable
```

Une variable peut être déclarée à de nombreux endroits dans le programme, l'emplacement de sa déclaration détermine quelles parties du programme auront accès à cette variable.

☞ **Les types de données**

- **byte**

Byte permet de stocker une valeur entière sur 8 bits (pas de chiffre à virgule donc). Cela permet d'aller de 0 à 255.

```
byte someVariable = 190; //déclare "someVariable" en tant que byte
```

- **int**

Int (pour integers, entier) est le type de base pour les chiffres (sans virgule) des variables. Le stockage se fait sur 16 bits, permettant d'aller de 32 767 à - 32 768.

```
int someVariable = 1500; //déclare "someVariable" en tant que int
```

- **long**

Il s'agit d'un type de variable de grande taille : la donnée (chiffre sans virgule) est stockée sur 32 bits. Cela permet d'aller de 2 147 483 647 à - 2 147 483 648.

```
long someVariable = 90000; //déclare "someVariable" en tant que long
```

- **float**

Ce type permet de stocker les chiffre "flottant" ou chiffre à virgule. Le float bénéfice d'une meilleure résolution que les entiers (int) : ils sont stockés sur 32 bits, soit de 3,4028235^38 à -3,4028235^38. **Note :** Les chiffres flottants ne sont pas exacts et il arrive que des résultats étranges ressortent de leur comparaison.

```
float someVariable = 3,14; //déclare "someVariable" en tant que chiffre flottant
```

- **bool**

Le type bool prend 2 valeurs : Vrai (true) ou Faux (false). Il est stocké sur 8 bits et se comporte ainsi :

0 = FAUX = FALSE
1 = VRAI = TRUE

```
bool someVariable = false; //la valeur est 0 donc false (faux)
```

- **char**

Le type char est utilisé pour stocker une lettre. Le stockage se fait sur 8 bits et permet de stocker une seule lettre. **Note :** Pour assigner une valeur à la variable char, on l'écrira entre '.

```
char someVariable = 'A'; //déclare someVariable en tant que lettre
```

- **string**

Il ne s'agit pas exactement d'un type de variable, il s'agit d'une classe. La classe String est comparable a une variable possédant ses propres fonctions. String permet de stocker une chaîne de caractères, comme pourrait le faire un tableau de caractère. String permet de réaliser des opérations sur les chaînes de caractères plus complexes que ce qui se ferait sur un tableau de caractères.

```
String someVariable = "Aux sciences, citoyens"; //someVariable contient la chaîne de caractères
```

☞ Les opérations arithmétiques

Les opérations d'arithmétique incluent les additions, les soustractions, les multiplications et les divisions. Elles retournent respectivement la somme, la différence, le produit et le quotient des 2 opérants.

L'opération est réalisée en utilisant les types de données des opérants, de façon à ce que, par exemple, 9 / 4 donnera 2 plutôt que 2,25 car 9 et 4 sont des entiers (int), or le type "int" ne peut être un chiffre à virgule. Cela veut aussi dire que le résultat peut être trop grand pour le type de variable initiale.

```
y = y + 3;
x = x - 7;
r = r / 5;
i = j * 6;
```

☞ Les opérations d'incrémentation

Les opérateurs d'incrémentation permettent de réaliser une opération simple, ils sont communément utilisés dans la boucle for, voir plus bas. Il en existe 2 : l'incrémentation et la décrémentation.

```
x ++ //équivaut à x = x + 1 ou d'incrémenter x par 1
x -- //équivaut à x = x - 1 ou de décrémenter x par 1
```

☞ Les opérations de comparaison

La comparaison d'une variable ou d'une constante avec une autre est souvent utilisée afin de tester si une condition spécifique est vraie ou non. Dans les exemples suivants, l'opérateur est utilisé pour indiqué une condition :

```
x == y //x est égal à y
x != y //x n'est pas égal à y
x < y //x est plus petit que y
x > y //x est plus grand que y
x <= y //x est plus petit ou égal à y
x >= y //x est plus grand ou égal à y
```

☞ Les opérateurs logiques

Les opérateurs logiques sont utilisés pour comparer deux expressions/conditions. Ils renvoient TRUE (vrai) ou FALSE (faux) selon l'opération. Il y a trois opérateurs logiques : AND (et), OR (ou) et NOT (contraire), ils sont souvent utilisés dans les if (voir plus bas) :

AND
```
if (x > 0 && x < 5) //vrai seulement si les deux conditions sont vraies
```

OR
```
if (x > 0 || y > 0) //vrai si une des condition est vraie
```

NOT
```
if (!x > 0) //vrai seulement si la condition est fausse
```

☞ **Les constantes**

Le langage Arduino possède des valeurs prédéfinies; elles sont appelée constantes. Elles sont utilisées pour rendre un programme plus simple à lire. Il en existe différents groupes.

- **Vrai / Faux (True / False)**

Il s'agit de variable du type booléen. FALSE signifie que la constante est à 0 alors que TRUE désigne l'état différent de 0 : cela peut être 1 (par convention) mais cela peut aussi être -1, 2, -200, ...

```
if (b == TRUE)
{
……….;
}
```

- **Haut / Bas (High / Low)**

Il s'agit des constantes utilisées pour lire ou pour écrire sur les broches, soit le niveau HIGH (ON, 1, 5 volts) soit le niveau LOW (OFF, 0, 0 volts).

```
digitalWrite(13,HIGH);
```

- **Entrée / Sortie (Input / Output)**

Ces constantes sont utilisées avec la fonction pinMode(), elles définissent le type de broche digitale :
INPUT pour entrée
OUTPUT pour sortie

```
pinMode(13, OUTPUT);
pinMode(12, INPUT);
```

☞ **Les structures de contrôle**

- **if**

La structure de test "if" (si) vérifie si une certaine condition est atteinte, par exemple si une valeur est égale à un certain nombre, et exécute des instructions contenu dans la structure. Si la condition est fausse, le programme quitte la structure et reprend à la ligne suivante. Le format du if est :

```
if( x == 10 )
{
Faire quelque chose
}
```

- **if .... else**

if... else (si...sinon) est utilisé pour dire "sinon fait...". Par exemple, si on veut tester une entrée digitale et faire ne action si elle vaut HIGH et d'autres actions si elle vaut LOW, on écrira :

```
if (inputPin == HIGH)
{
doThingA;
}
else
{
 doThingB;
}
```

- **Switch ... case**

La structure switch... case (commutateur de cas) permet de dresser une liste de tests ou de cas (comme le "if") et d'exécuter le code correspondant si un des cas du test est vrai.

La principale différence entre le switch... case et le if... else est que le switch... case permet de continuer les tests même si un des cas est vrai, contrairement au if...else qui quittera dans le même cas.

```
switch (someVariable)
{
case A : doSomething;
case B : doOtherSomething;
 }
```

someVariable sera comparée à A, si le test est vrai, doSomething sera exécuté. Une fois le code exécuté (ou si le test est faux), "someVariable" sera comparé à B et ainsi de suite. Dans le cas où on voudrait s'arrêter à un des cas, on ajoutera l'instruction *break;*. De plus, si aucun des cas n'est vérifié, on peut ajouter le cas *default* qui sera exécuté si aucun test n'est vrai.

```
switch (someVariable) {
case A : doSomethingA; //si someVariable vaut A, doSomethingA est exécuté
        break; //une fois le code exécuté, on sort de la structure
case B : doSomethingB; //en l'absence de "break", si someVariable est égale à B puis à C
case C: doSomethingC; //doSomethingB et doSomethingC seront exécutés
default : doSomethingByDefault //dans le cas où aucun cas ne serait vérifié,
                    //doSomethingByDefault sera exécuté puis on sortira de la structure
}
```

- **for**

La structure for (pour) est utilisée afin de répéter une série d'instructions (comprise entre les accolades) le nombre de fois voulu. Un compteur à incrémenter est souvent utilisé afin de sortir de la boucle une fois le nombre de tour fait. Cette structure est composée de trois parties séparées par un point-virgule (;) dans l'entête de la boucle :

```
for (initialisation; condition; expression d'incrémentation) {
doSomething;
}
```

L'initialisation d'une variable locale, ou compteur d'incrémentation, permet d'avoir une variable ne servant qu'une fois dans le programme.

A chaque tour de boucle, la condition de l'entête est testée. Si cette condition est vraie, doSomething est exécuté ainsi que l'expression d'incrémentation et la condition est de nouveau testée. Quand la condition devient fausse, la boucle prend fin.

L'exemple suivant commande avec un entier i à 0, on vérifie s'il est inférieur à 20 et si c'est vrai, on l'incrémente de 1 et on exécute le programme contenu dans les accolades :

```
for (int i = 0; i<20; i++) {
 digitalWrite(13, HIGH);
delay(250);
digitalWrite(13,LOW);
delay(250);
}
```

- **while**

La boucle "while" (tant que) continuera infiniment tant que l'expression entre parenthèse est vraie. Quelque chose doit changer afin de modifier la condition sinon le programme ne sortira jamais de la boucle while. Cela peut être dans le code, comme l'incrémentation d'une variable, ou une condition extérieure, comme le test d'un capteur.

L'exemple suivant test si "someVariable" est inférieur à 200 et si c'est vrai, le code entre les accolades est exécuté et la boucle continue tant que "someVariable" est inférieure à 200.

```
while (someVariable < 200) //test si "someVariable" est inférieur à 200
{
doSomething; //exécution du code
someVariable++ //incrémentation de "someVariable" par 1
 }
```

- **do ... while**

La boucle do... while (faire... tant que) fonctionne de la même façon que la boucle while, à la seule exception que la condition est testée à la fin de la boucle, de ce fait le code dans la boucle s'exécute toujours au moins une fois

L'exemple suivant assigne à la variable "x" la valeur lireCapteur(), puis fait une pause de 50ms et continue infiniment tant que x est plus petit que 100.

```
do {
 x = lireCapteur(); //assignation de la valeur de lireCapteur à x
delay(50); //pause durant 50 milli-secondes
 } while (x < 100); //on recommence la boucle si x est inférieur à 100
```

☞ **Les entrées et sorties numériques**

- **pinMode(pin, mode)**

Utilisée dans le void setup(), cette fonction permet de configurer une broche soit en entrée (INPUT) soit en sortie (OUTPUT).

```
pinMode(broche, OUTPUT); //fixe la broche en sortie
```

- **digitalRead(pin)**

Cette fonction permet de lire la valeur à la broche digitale indiquée. La valeur lue étant soit HIGH soit LOW. La broche ciblée ne peut être qu'une variable ou une constante comprise entre 0 et 13 (pour les Arduino Uno).

```
valeur = digitalRead(broche); //assigne à "valeur" la valeur lue à la broche
```

- **digitalWrite(pin, value)**

Il s'agit du contraire de digitalRead() : cette fonction permet de fixer une sortie digitale soit au niveau HIGH soit au niveau LOW. Cette sortie est comprise entre 0 et 13 sur Arduino Uno et doit être spécifiée soit par une variable soit par une constante.

```
digitalWrite(broche, HIGH) //fixe la sortie "broche" à HIGH
```

☞ **Les entrées et sorties analogiques**

- **analogRead(pin)**

Cette fonction permet d'aller lire la valeur sur une sortie analogique avec 10 bits de résolution. Elle ne fonctionne que sur les broches analogiques (0-5) et renvoie un résultat compris entre 0 et 1023.

```
valeur = analogRead(broche); //assigne à "valeur" ce qui est lue sur la broche
```

Note : Les broches analogiques ne sont pas des broches digitales, elle ne nécessite pas d'être déclarée comme des sorties ou des entrées.

- **analogWrite(pin, value)**

analogWrite permet d'envoyer un pseudo-signal analogique en utilisant le matériel via les broches possédant un "pulse width modulation" (PWM ou modulation de largeur d'impulsions). Sur les Arduino les plus récents (équipé du contrôleur ATmega168), cette fonctionnalité se trouve sur les broches 3, 5, 6, 9, 10 et 11. Sur les plus anciennes (ATmega8), seules les broches 9, 10 et 11 étaient équipées. Ce signal est envoyé sous forme de valeur, comprise entre 0 et 255.

```
analogWrite(broche, valeur); //envoie "valeur" sur la "broche" analogique
```

☞ **La gestion du temps**

- **delay(ms)**

delay permet de marquer une pause dans le programme, le temps est en millisecondes

```
delay(1000); //pause d'une seconde
```

- **millis()**

Cette fonction renvoie le nombre de millisecondes écoulé depuis que l'Arduino a commencé à exécuté le programme actuel sous l'état d'un unsigned long.

```
valeur = millis(); //assigne à "valeur" le nombre de milli-secondes écoulé
```

Note : Le nombre est remis à 0 (par dépassement de tailles de variables) au bout d'environ 9 heures.

☞ **La communication Série**

- **serial.begin(débit)**

Permet le port série et donne la vitesse de transmission (ou débit, en bits par seconde ou bauds). La vitesse usuelle est de 9600 bauds avec les ordinateurs mais d'autres vitesses sont supportées.

```
void setup() {
Serial.begin(9600); //Ouvre le port série et fixe le débit à 9600 bauds
}
```

Note : Lorsque la liaison série est utilisée, les broches digitales 0 (RX) et 1 (TX) ne peuvent être utilisées en même temps.

- **serial.println(données)**

Permet d'envoyer des données sur le port série et d'automatiquement retourner à la ligne ensuite. La commande Serial.print() fonctionne de la même façon mais sans le retour à la ligne, pouvant être moins facile à lire via le moniteur série.

```
void setup() {
Serial.begin(9600); //affecte le débit à 9600 bauds
}
void loop() {
 Serial.println(analogRead(0)); //envoi la valeur analogique
delay(1000); //marque une pause d'une seconde
}
```

5.3. La bibliothèque

Une bibliothèque est un ensemble de fonctions qui s'ajoutent aux fonctions de base du logiciel de l'Arduino et qui adressent plus particulièrement un domaine spécifique de la programmation comme, par exemple :

- EEPROM - lire et écrire dans cette zone de mémoire permanente du micro-contrôleur ;
- Ethernet - accéder à un réseau TCP/IP comme Internet, mais ça peut être aussi un réseau local à votre domicile, avec l'aide de la carte Ethernet ;
- LiquidCrystal - utiliser un écran LCD ;
- SD - lire et écrire une carte SD de technologie flash et formatée en FAT16 ou FAT32, sauf reformatage de votre part, toutes les cartes SD le sont ;
- Servo - piloter des servomécanismes ;
- WiFi - se connecter à un réseau sans fil à partir d'une carte WiFi ;
- Wire - Communiquer entre Arduino et avec des périphériques capteurs, actionneurs avec le protocole I2C.

A noter que ces bibliothèques sont **installées en même temps que l'IDE**. Vous n'aurez pas à les ajouter. Mais surtout, il existe des multitudes de bibliothèques dans pratiquement tous les domaines, ce qui donne à l'Arduino cette puissance incontestable et son succès.

Ce paragraphe concerne donc l'installation de librairies supplémentaires qui n'y sont pas au départ.

☞ **Bibliothèque ou Librairie ?**

Avant de commencer, revenons sur un point de traduction. Arduino est un projet où l'anglais prédomine. *Library* est normalement traduit littéralement en français par *bibliothèque*. Cependant l'anglicisme a pris le dessus et vous trouverez beaucoup de pages en français où on parle de *librairie* Arduino.
Nous allons voir dans cet article comment installer une bibliothèque qui n'est pas de base dans l'IDE Arduino.

☞ **Où les trouver ?**

Comme Arduino est un projet communautaire, vous trouverez des bibliothèques sur le web en général.
Quelques bibliothèques sont recensées sur le site officiel . Si vous ne trouvez pas votre bonheur, une recherche avec le type de **shield** ou de composant accompagné de **Arduino** et **library** vous donnera les pointeurs souhaités.

☞ **Où les installer ?**

C:\Users\nom du PC\Documents\Arduino\libraries

Vous voyez que les bibliothèques officielles ne sont pas dans ce répertoire. Afin que chaque utilisateur de l'ordinateur y ait accès, elles sont installées dans un répertoire qui diffère du précédent et même selon les ordinateurs :

- sous Windows 32 bits, dans C :\Program Files\Arduino\libraries ;
- sous Windows 64 bits, dans C :\Program Files (x86)\Arduino\libraries ;

Vous pourrez faire une installation ici si vous en avez les droits pour faire profiter tous les utilisateurs de l'ordinateur de cette bibliothèque. Toutefois, ce n'est pas une procédure prévue dans l'IDE Arduino et c'est inutile si vous êtes le seul à programmer sur Arduino avec votre ordinateur.

☞ **Méthode d'installation**

Dans l'IDE Arduino, vous avez le menu Croquis/Inclure une bibliothèque/Ajouter la bibliothèque .ZIP... Vous sélectionnez l'archive de la bibliothèque et vous cliquez sur OK. Elle est installée.

Redémarrez l'IDE pour profiter de la bibliothèque.

Depuis les récentes versions de l'IDE Arduino, un Gestionnaire de bibliothèques est disponible par la commande Croquis/Inclure une bibliothèque/Gérer les bibliothèques... Il permet de retrouver une liste détaillée des bibliothèques présentes, de les mettre à jour lorsqu'une évolution a été détectée chez son créateur, et d'en installer de nouvelles proposées automatiquement. Cette liste peut être triée par thème, ou par état (installées, à mettre à jour...)

☞ **Désinstallation**

Supprimez simplement le répertoire de la bibliothèque et redémarrez l'IDE pour que ça soit pris en compte.

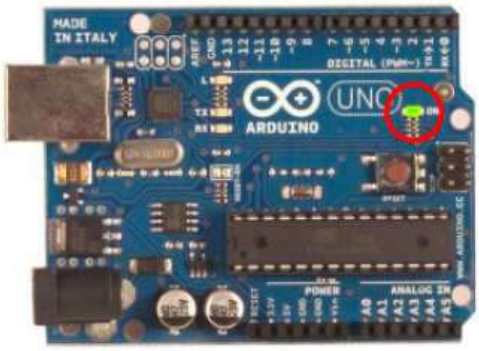
6. Atelier - Travaux pratiques

6.1. TP 1 : Test de fonctionnement de la carte

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 !

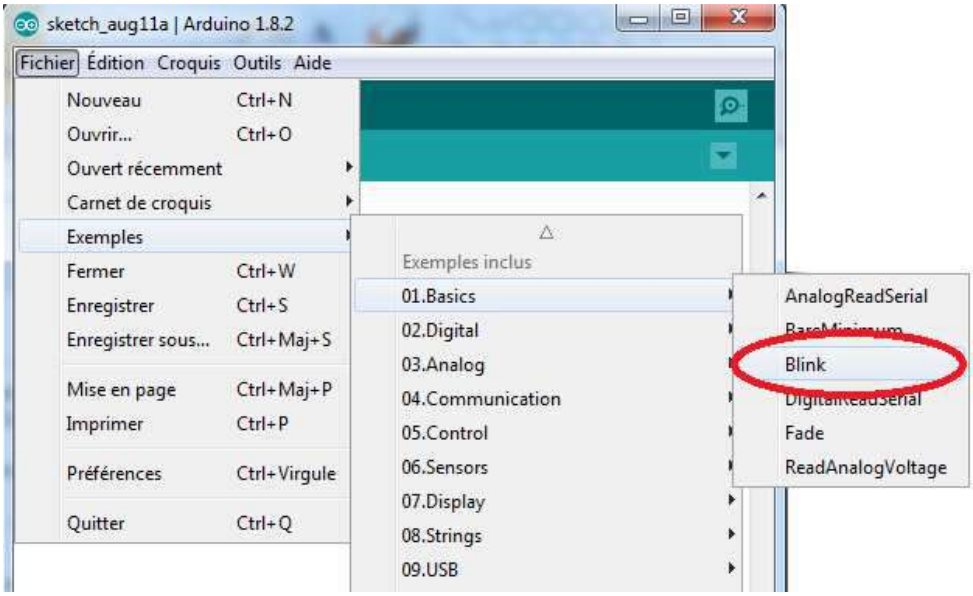
Nous allons tester notre matériel en chargeant un programme qui fonctionne dans la carte.



Carte connectée et alimentée

1ère étape : ouvrir un programme

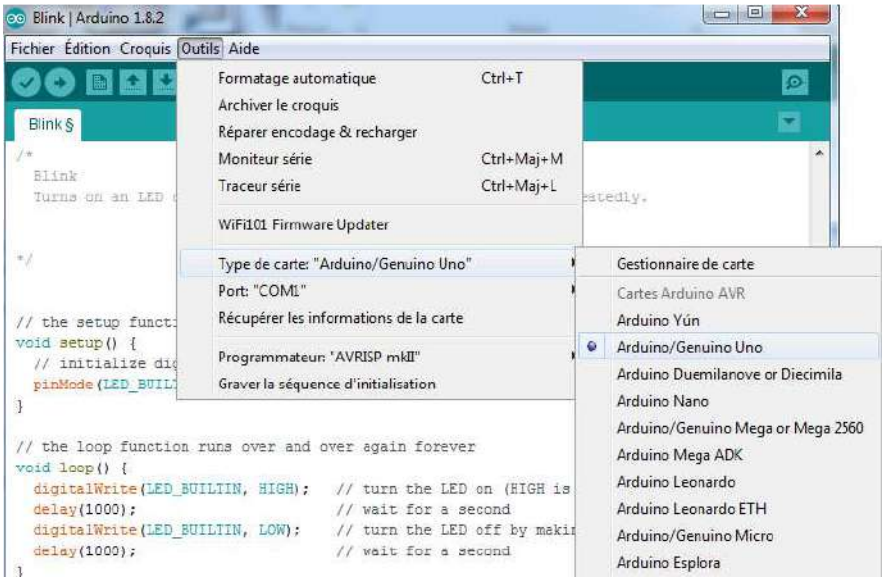
Nous allons choisir un exemple tout simple qui consiste à faire clignoter une LED. Son nom est *Blink* et vous le trouverez dans la catégorie *Basics* :



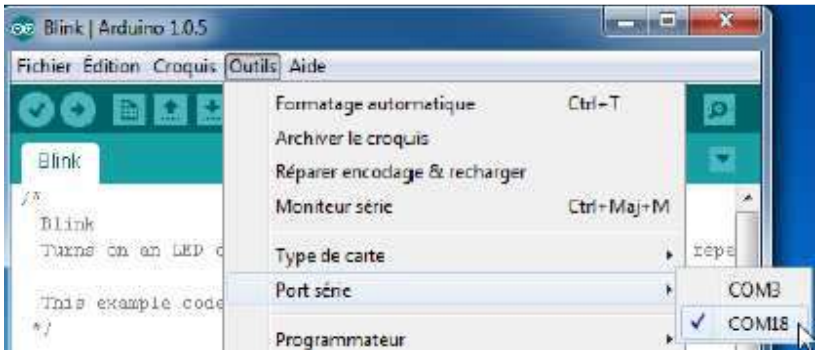
Une fois que vous avez cliqué sur *Blink*, une nouvelle fenêtre va apparaître. Elle va contenir le programme *Blink*. Vous pouvez fermer l'ancienne fenêtre qui va ne nous servir plus à rien.

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

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 "Arduino Uno" qui est coché. Si ce n'est pas le cas, cochez-le.



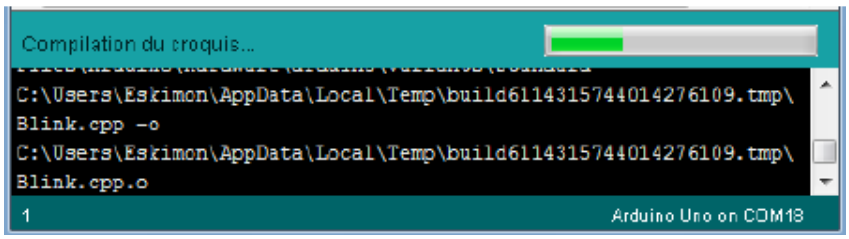
***Ensuite**, choisissez le port de connexion de la carte.* Allez dans le menu *Tools*, puis *Serial port*. Là, vous choisissez le port COM X, X étant le numéro du port qui est affiché. Ne choisissez pas COM1 car il n'est quasiment jamais connecté à la carte. Dans mon cas, il s'agit de COM5 :



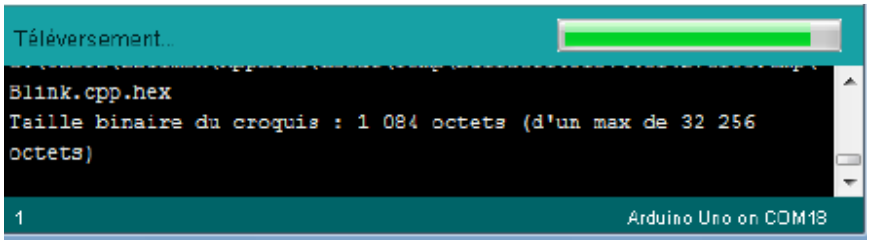
Pour la dernière étape, il va falloir envoyer le programme dans la carte. Pour ce faire, il suffit de cliquer sur le bouton *Téléverser*, en blanc sur la photo :

- ***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 :

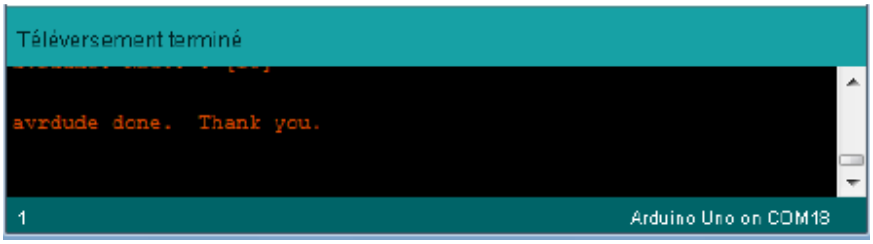


En bas dans l'image, vous voyez le texte : "*Téléversement…*", cela signifie que le logiciel est en train d'envoyer le programme dans la carte. Une fois qu'il a fini, il affiche un autre message :



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!



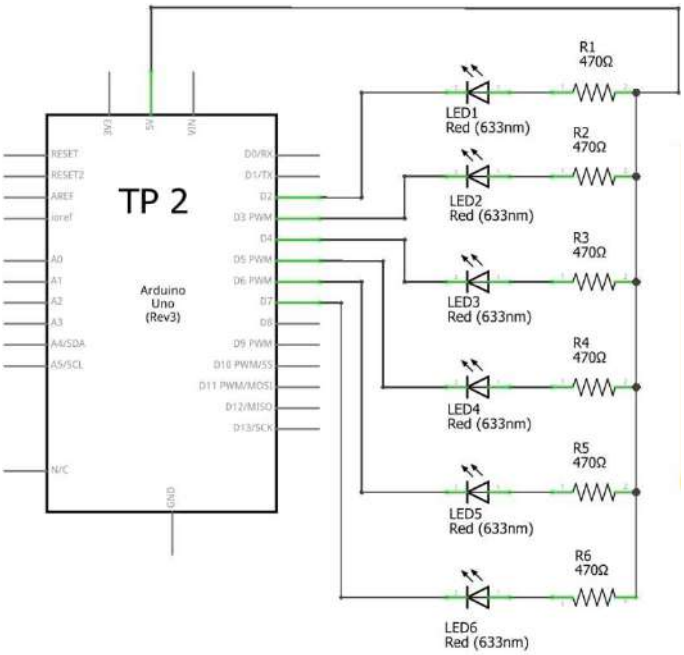
6.2. TP 2 : Allumage de LED(s)

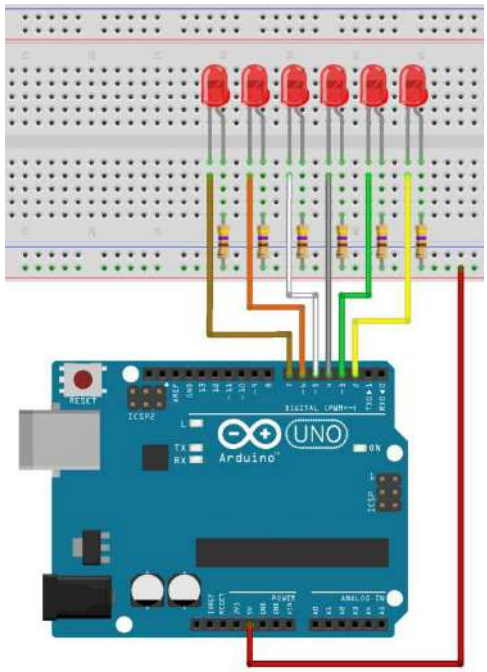
☞ ***But :***

Allumer une diode LED avec plusieurs variantes :

a. Allumer une LED
b. Faire clignoter une LED
c. Faire clignoter 2 LED
d. Faire clignoter 3 LED
e. Chenillard 6 LED
f. BONUS : Créer une fonction paramétrée "clignotantLED"
(paramètres : n° broche et durée (ms))

☞ ***Schéma électrique :***





☞ ***Code :***

☞ ***Solutions :***

- ***variante "b"***

```
void setup()
{
pinMode(2, OUTPUT);
// met la pin numérique en sortie
}
void loop()
{
digitalWrite(2, LOW); // allume la LED
delay(1000); // attend une seconde
digitalWrite(2, HIGH); // éteint la LED
delay(1000); // attend une seconde
}
```

- ***variante "c"***

```
void setup(){
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
}

void loop()
{
  digitalWrite(2, LOW);
  digitalWrite(3, HIGH);
  delay(500);
  digitalWrite(2, HIGH);
  digitalWrite(3,LOW);
  delay(500);
}
```

- ***variante "d"***

```
int L1 =  2;
int L2 =  3;
int L3 =  4;

void setup()   {

  pinMode(L1, OUTPUT);
  pinMode(L2, OUTPUT);
  pinMode(L3, OUTPUT);
}

void loop()
{
  digitalWrite(L1, LOW);
  delay(500);
  digitalWrite(L2, LOW);
  delay(500);
  digitalWrite(L3, LOW);
  delay(500);
  digitalWrite(L1, HIGH);
  digitalWrite(L2, HIGH);
  digitalWrite(L3, HIGH);
  delay(500);
}
```

- ***variante "e"***

```
void setup()
{
pinMode(2, OUTPUT); // LED D1
pinMode(3, OUTPUT); // LED D2
pinMode(4, OUTPUT); // LED D3
pinMode(5, OUTPUT); // LED D4
pinMode(6, OUTPUT);  // LED D5
pinMode(7, OUTPUT);  // LED D6
}

void loop() {

 delay(200);
 digitalWrite(7, HIGH);
 digitalWrite(2, LOW);

  delay(200);
 digitalWrite(2,HIGH);
 digitalWrite(3,LOW);

  delay(200);
 digitalWrite(3,HIGH);
 digitalWrite(4,LOW);

 delay(200);
 digitalWrite(4,HIGH);
 digitalWrite(5,LOW);

 delay(200);
 digitalWrite(5,HIGH);
 digitalWrite(6,LOW);

 delay(200);
 digitalWrite(6,HIGH);
 digitalWrite(7,LOW);
 }
```

- ***Bonus : variante "f"***

=> Voir professeur

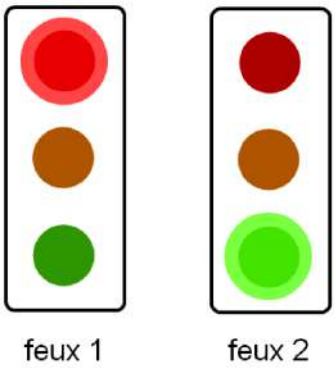
6.3. TP 3 : Feux routiers

☞ ***But :***

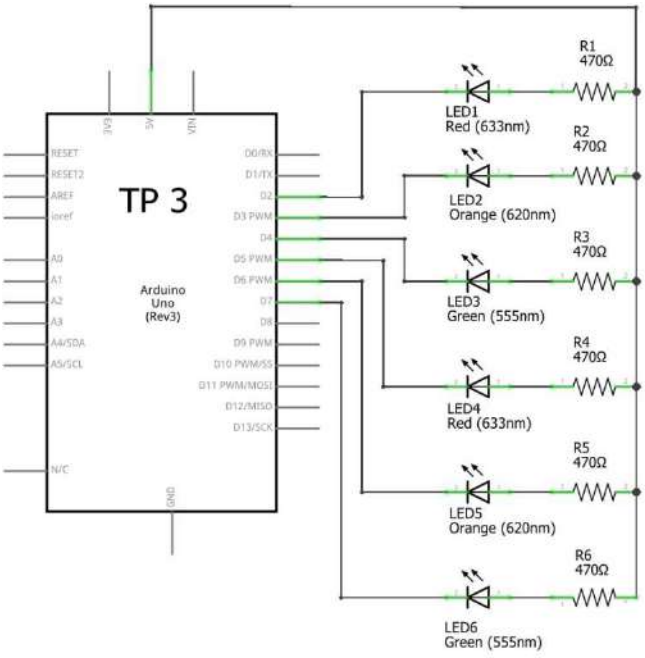
Réaliser un feu de signalisation routier.

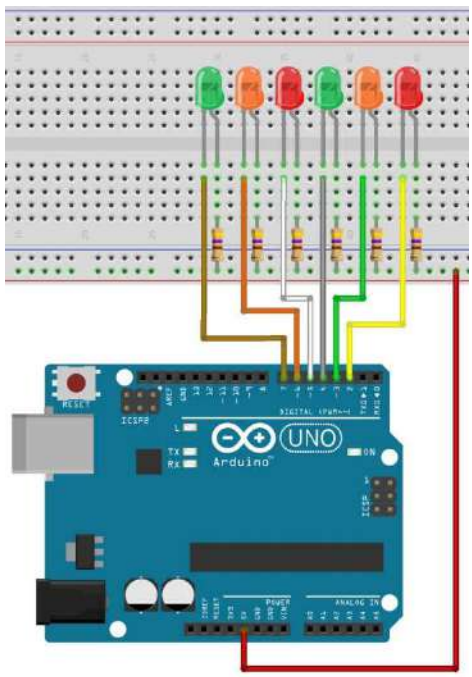
Pour info, le feu rouge dure 3 secondes et le feu orange dure 1 seconde

(conseil : le TP 2 vous aidera pour la réalisation de cet atelier)



☞ ***Schéma électrique :***





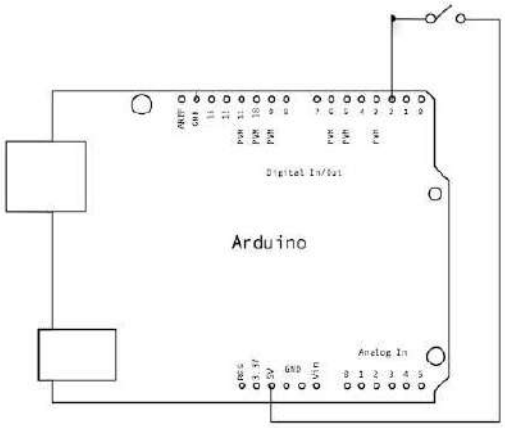
☞ ***Solution :***

```
//définition des broches
const int led_rouge_feux_1 = 2;
const int led_jaune_feux_1 = 3;
const int led_verte_feux_1 = 4;
const int led_rouge_feux_2 = 5;
const int led_jaune_feux_2 = 6;
const int led_verte_feux_2 = 7;
void setup()
{
//initialisation en sortie de toutes les broches
pinMode(led_rouge_feux_1, OUTPUT);
pinMode(led_jaune_feux_1, OUTPUT);
pinMode(led_verte_feux_1, OUTPUT);
pinMode(led_rouge_feux_2, OUTPUT);
pinMode(led_jaune_feux_2, OUTPUT);
pinMode(led_verte_feux_2, OUTPUT);
//on initialise toutes les LED éteintes au début du programme (sauf les deux feux rouges)
digitalWrite(led_rouge_feux_1, LOW);
digitalWrite(led_jaune_feux_1, HIGH);
digitalWrite(led_verte_feux_1, HIGH);
digitalWrite(led_rouge_feux_2, LOW);
digitalWrite(led_jaune_feux_2, HIGH);
digitalWrite(led_verte_feux_2, HIGH);
}

void loop()
{
// première séquence
digitalWrite(led_rouge_feux_1, HIGH);
digitalWrite(led_verte_feux_1, LOW);
delay(3000);
// deuxième séquence
digitalWrite(led_verte_feux_1, HIGH);
digitalWrite(led_jaune_feux_1, LOW);
delay(1000);
// troisième séquence
digitalWrite(led_jaune_feux_1, HIGH);
digitalWrite(led_rouge_feux_1, LOW);
delay(1000);
/* deuxième partie du programme, on s'occupe du feux numéro 2 */
// première séquence
digitalWrite(led_rouge_feux_2, HIGH);
digitalWrite(led_verte_feux_2, LOW);
delay(3000);
// deuxième séquence
digitalWrite(led_verte_feux_2, HIGH);
digitalWrite(led_jaune_feux_2, LOW);
delay(1000);
// deuxième séquence
digitalWrite(led_jaune_feux_2, HIGH);
digitalWrite(led_rouge_feux_2, LOW);
delay(1000);
// le programme va reboucler et revenir au début
}
```

6.4. TP 4 : Boutons poussoirs et LED

☞ ***Avant de commencer, ce qu'il faut savoir sur les boutons !***

Pour câbler un bouton poussoir sur la carte Arduino, on pourrait penser qu'il suffit de le connecter comme suit



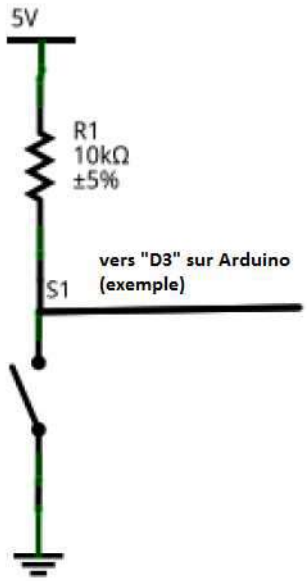
Lorsque le bouton est pressé, l'entrée est à 5 V, mais lorsque le bouton n'est pas pressé, elle n'est pas définie car elle est "en l'air".

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(...).

Lorsque l'on fait de l'électronique, on a toujours peur des perturbations (générées par plein de choses : des lampes à proximité, un téléphone portable, un doigt sur le circuit, l'électricité statique, …). On appelle ça des contraintes de **CEM**.

Ces perturbations sont souvent inoffensives, mais perturbent beaucoup les montages électroniques. Il est alors nécessaire d'en prendre compte lorsque l'on fait de l'électronique de signal. Par exemple, dans certains cas on peut se retrouver avec un bit de signal qui vaut 1 à la place de 0, les données reçues sont donc fausses.



Pour contrer ces effets nuisibles, on place en série avec le bouton une résistance de pull-up. Cette résistance sert à "tirer" ("to pull" in english) le potentiel vers le haut (up) afin d'avoir un signal clair sur la broche étudiée.

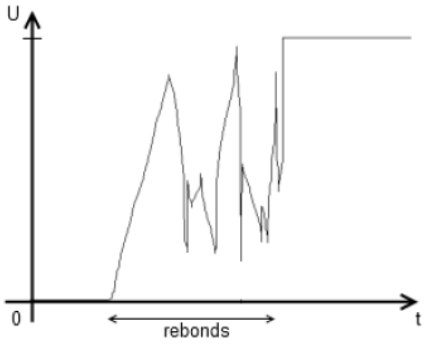
Sur le schéma suivant, on voit ainsi qu'en temps normal le "signal" à un potentiel de 5V. Ensuite, lorsque l'utilisateur appuiera sur le bouton une connexion sera faite avec la masse. On lira alors une valeur de 0V pour le signal. Voici donc un deuxième intérêt de la résistance de pull-up, éviter le court-circuit qui serait généré à l'appui.

Que peux-tu dire sur la résistance de pull-down ?

<u>=> Filtrer les rebonds</u>

Les boutons ne sont pas des systèmes mécaniques parfaits. Du coup, lorsqu'un appui est fait dessus, le signal ne passe pas immédiatement et proprement de 5V à 0V.

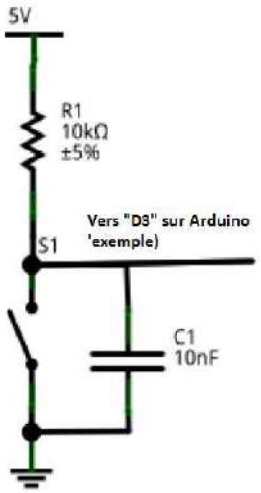
En l'espace de quelques millisecondes, le signal va "sauter" entre 5V et 0V plusieurs fois avant de se stabiliser. Il se passe le même phénomène lorsque l'utilisateur relâche le bouton.



Ce genre d'effet n'est pas désirable, car il peut engendrer des parasites au sein de votre programme (si vous voulez détecter un appui, les rebonds vont vous en générer une dizaine en quelques millisecondes, ce qui peut être très gênant dans le cas d'un compteur par exemple).

Voilà, ci-contre, un exemple de chronogramme relevé lors du relâchement d'un bouton poussoir.

Pour atténuer ce phénomène, nous allons utiliser un condensateur en parallèle avec le bouton.



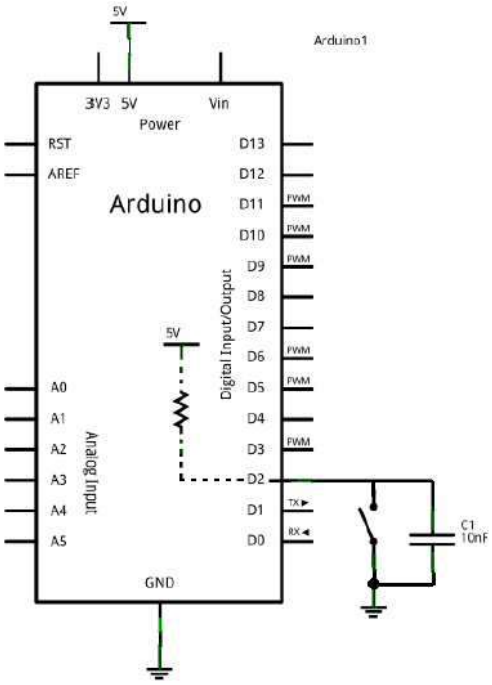
Ce composant servira ici "d'amortisseur" qui absorbera les rebonds (comme sur une voiture avec les cahots de la route). Le condensateur, initialement chargé, va se décharger lors de l'appui sur le bouton.

S'il y a des rebonds, ils seront encaissés par le condensateur durant cette décharge. Il se passera le phénomène inverse (charge du condensateur) lors du relâchement du bouton.

Ce principe est illustré à la figure ci-contre :

=> Résistance interne de pull-up

Il faut savoir qu'il existe des résistances de pull-up en interne du microcontrôleur de l'Arduino, ce qui évite d'avoir à les rajouter par nous mêmes par la suite.



Ces dernières ont une valeur de 20 kiloOhms. Elles peuvent être utilisés sans aucune contraintes techniques.

Cependant, si vous les mettez en marche, il faut se souvenir que cela équivaut à mettre la broche à l'état haut (et en entrée évidemment).

Donc si vous repassez à un état de sortie ensuite, rappelez vous bien que tant que vous ne l'avez pas changée elle sera à l'état haut.

Un TP mettra en lumière la marche à suivre pour activer ces résistances internes.

=> Remarque importante :

Attention, pour le pin 13, il est déconseillé d'utiliser le mode INPUT_PULLUP. Si vous devez vous servir du pin 13 comme pin de lecture, préférez un montage avec une résistance externe en pull-up ou pull-down. L'explication se trouve dans le fait que le pin 13 est aussi lié à une LED et une résistance. Il ne fournira donc pas du +5V, mais du +1.7V à cause de la LED et de la résistance en série qui baissent la tension. De ce fait la lecture sera toujours à LOW.

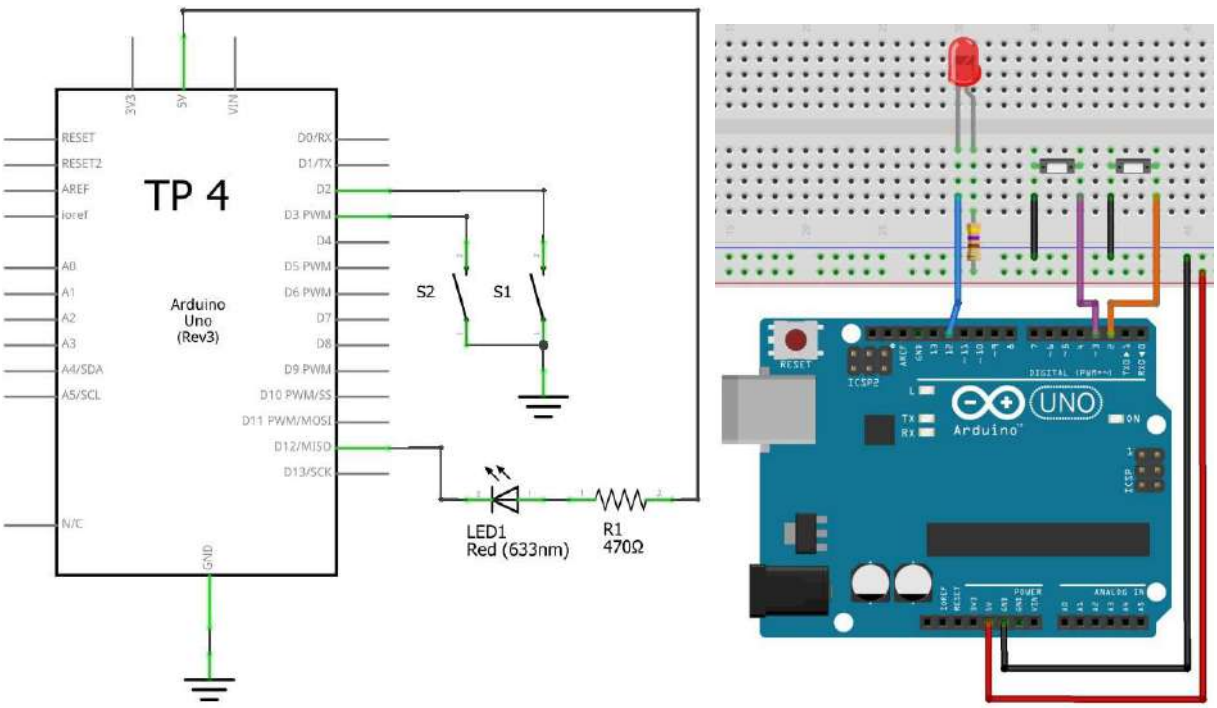
☞ ***But :***

Allumer la LED en fonction de l'état du bouton. Cet exercice comporte plusieurs variantes :

a. Allumer une LED avec un bouton poussoir
b. Allumer une LED avec deux BP en fonction logique OU
c. Allumer une LED avec deux BP en fonction logique ET
d. Allumer une LED avec un BP et l'éteindre avec l'autre BP
e. Lire les états des boutons sur le port série

Remarque : On utilisera les résistances de pull-up internes de l'Arduino et on ne prendre pas de précautions par rapport aux possibles rebonds des boutons !

☞ ***Schéma électrique :***



☞ ***Solution :***

- ***variante "a"***

```
//le bouton connecté à la broche 2 de la carte Adruino
const int bouton = 2;
//la LED à la broche 12
const int led = 12;
//variable qui enregistre l'état du bouton
int etatBouton
void setup()
{
pinMode(led, OUTPUT); //la led est une sortie
pinMode(bouton, INPUT); //le bouton est une entrée
etatBouton = HIGH; //on initialise l'état du bouton
comme "relaché"
}
void loop()
{
etatBouton = digitalRead(bouton); // bouton = 2
if(etatBouton == HIGH) //test si le bouton a un
niveau logique HAUT
{
digitalWrite(led,HIGH); //la LED reste éteinte
}
else //test si le bouton a un niveau logique BAS
{
digitalWrite(led,LOW); //la LED est allumée
}
}
```

- ***variante "b"***

```
void setup() {
 pinMode(2, INPUT_PULLUP); //mode lecture pour le
bouton avec résistance interne pullup
 pinMode(3, INPUT_PULLUP); //mode lecture pour le
bouton avec résistance interne pullup
 pinMode(12, OUTPUT);
}
void loop() {
 boolean etatS1 = digitalRead(2);
//lecture bouton S1 et stockage dans etatS1
 boolean etatS2 = digitalRead(3);
//lecture bouton S2 et stockage dans etatS2
 if ((etatS1==LOW)||(etatS2==LOW)) {
  digitalWrite(12,LOW); //Led allumée
 }
 else
{
 digitalWrite(12,HIGH);
}
}
```

- ***variante "c"***

```
void setup() {
 pinMode(2, INPUT_PULLUP); //mode lecture pour le
bouton avec résistance interne pullup
 pinMode(3, INPUT_PULLUP); //mode lecture pour le
bouton avec résistance interne pullup
 pinMode(12, OUTPUT);
}
void loop() {
 boolean etatS1 = digitalRead(2);
//lecture bouton S1 et stockage dans etatS1
 boolean etatS2 = digitalRead(3);
//lecture bouton S2 et stockage dans etatS2
 if ((etatS1==LOW)&&(etatS2==LOW)) {
  digitalWrite(12,LOW); //Led allumée
 }
 else
{
 digitalWrite(12,HIGH);
}
}
```

- ***variante "d"***

```
const int S1= 2;
 const int S2= 3;
const int led =  12;
int Etat_boutonS1 = 0;
 int Etat_boutonS2= 0;

// Configuration n'est utilisée qu'une seule fois
 void setup() {
  // Initialise les broches en entrées ou sorties:
  pinMode(led, OUTPUT);
 pinMode(S1, INPUT_PULLUP;
 pinMode(S2,INPUT_PULLUP);
 }
void loop(){
  // Lecture de l'etat logique des boutons
  Etat_boutonS1 = digitalRead(S1);
 Etat_boutonS2= digitalRead(S2);
    // Test logique des boutons poussoirs
  if (Etat_boutonS1 == LOW) {
   // Allume la LED
   digitalWrite(led, LOW);
 }
  if (Etat_bouton2 == LOW) {
   // Eteint la LED
   digitalWrite(led, HIGH);
 }
 }
```

- ***variante "e"***

=> vor professeur (en classe)

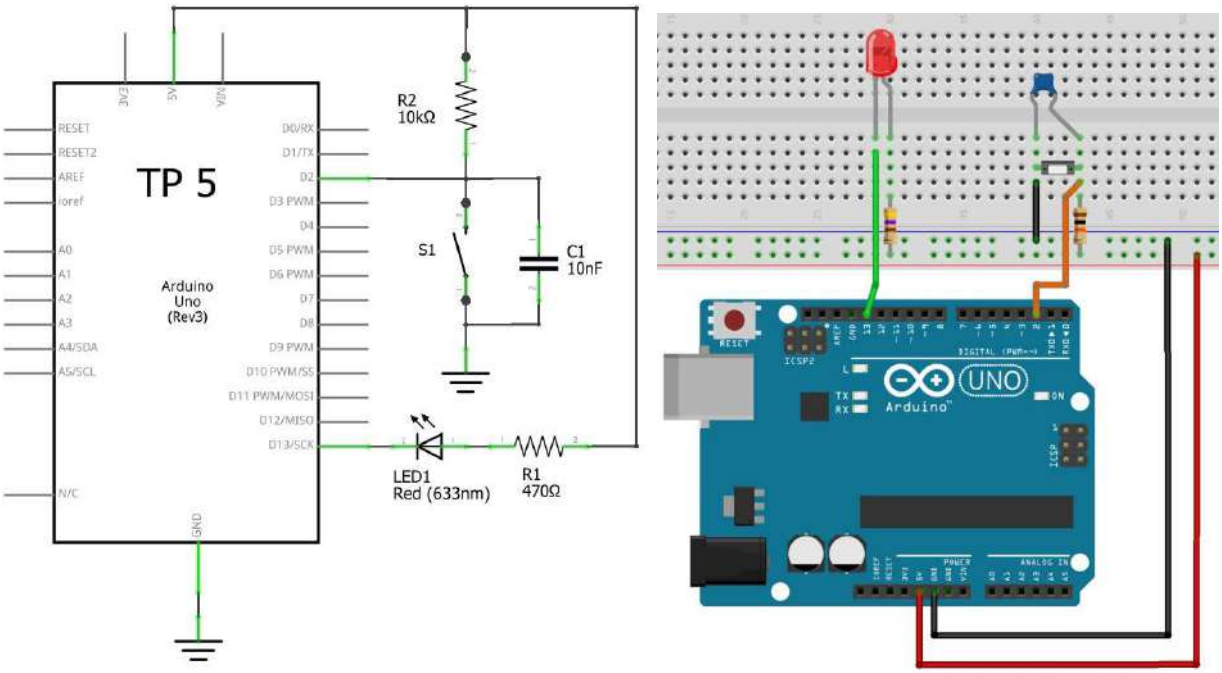
6.5. TP 5 : Bouton poussoir (avec précautions) et LED

☞ ***But :***

Allumer la LED en fonction de l'état du bouton. Le bouton est enfoncé, la LED s'allume. Le bouton est relâché, la LED s'éteint. On prendra les précautions nécessaires en ce qui concerne la résistance de pull-up ainsi que par rapport aux possibles rebonds du bouton poussoir !!

Attention remarque importante : Il est nécessaire de lire et de comprendre les explications données sur les résistances pull-up et pull-down ainsi que sur les précautions à prendre afin de limiter les rebonds !

☞ ***Schéma électrique :***



☞ ***Solution :***

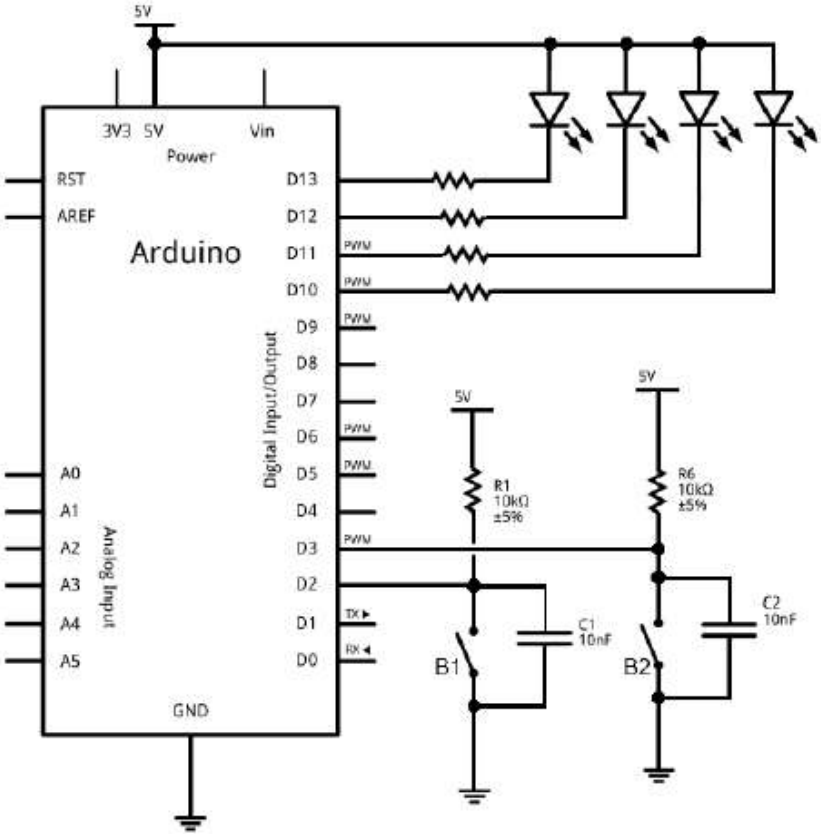
```
//le bouton est connecté à la broche 2 de la carte Adruino
const int bouton = 2;
//la LED à la broche 13
const int led = 13;
//variable qui enregistre l'état du bouton
int etatBouton

void setup()
{
pinMode(led, OUTPUT); //la led est une sortie
pinMode(bouton, INPUT); //le bouton est une entrée
etatBouton = HIGH; //on initialise l'état du bouton comme "relaché"
}


void loop()
{
etatBouton = digitalRead(bouton); //Rappel : bouton = 2
if(etatBouton == HIGH) //test si le bouton a un niveau logique HAUT
{
digitalWrite(led,HIGH); //la LED reste éteinte
}
else //test si le bouton a un niveau logique différent de HAUT (donc BAS)
{
digitalWrite(led,LOW); //le bouton est appuyé, la LED est allumée
}
}
```

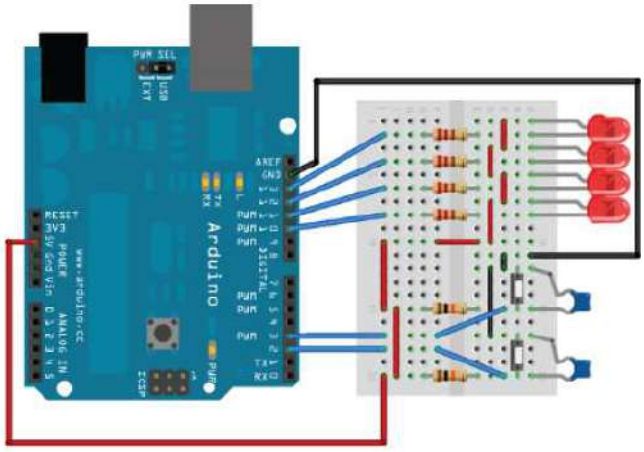
6.6. TP 6 : Bargraphe à 4 LEDs

☞ ***But :***

Réaliser un petit bargraphe avec 4 LED en fonction de 2 boutons poussoirs, un qui incrémentera le nombre de LED allumées, l'autre qui le décrémentera
Vous devrez utiliser une variable qui voit sa valeur augmenter ou diminuer entre 0 et 4 selon l'appui du bouton d'incrémentation ou de décrémentation.

☞ ***Schéma électrique :***





☞ ***Solution :***

```
/* déclaration des constantes pour les nom des broches ; ceci selon le schéma*/
const int btn_minus = 2;
const int btn_plus = 3;
const int led_0 = 10;
const int led_1 = 11;
const int led_2 = 12;
const int led_3 = 13;
/* déclaration des variables utilisées pour le comptage et le décomptage */
int nombre_led = 0; //le nombre qui sera incrémenté et décrémenté
int etat_bouton; //lecture de l'état des boutons (un seul à la fois mais une variable suffit)
int memoire_plus = HIGH; //état relâché par défaut
int memoire_minus = HIGH;
/* initilisation des broches en entrée/sortie */
void setup()
{
pinMode(btn_plus, INPUT);
pinMode(btn_minus, INPUT);
pinMode(led_0, OUTPUT);
pinMode(led_1, OUTPUT);
pinMode(led_2, OUTPUT);
pinMode(led_3, OUTPUT);
}
void loop()
{
//lecture de l'état du bouton d'incrémentation
etat_bouton = digitalRead(btn_plus);
//Si le bouton a un état différent que celui enregistré ET que cet état est "appuyé"
if((etat_bouton != memoire_plus) && (etat_bouton == LOW))
{
nombre_led++; //on incrémente la variable qui indique combien de LED devrons s'allumer
}
memoire_plus = etat_bouton; //on enregistre l'état du bouton pour le tour suivant

//et maintenant pareil pour le bouton qui décrémente
etat_bouton = digitalRead(btn_minus); //lecture de son état
//Si le bouton a un état différent que celui enregistré ET que cet état est "appuyé"
if((etat_bouton != memoire_minus) && (etat_bouton == LOW))
{
nombre_led;
//on décrémente la valeur de nombre_led
}
memoire_minus = etat_bouton; //on enregistre l'état du bouton pour le tour suivant
//on applique des limites au nombre pour ne pas dépasser 4 ou 0
if(nombre_led > 4)
{
nombre_led = 4;
}
if(nombre_led < 0)
{
nombre_led = 0;
}
//appel de la fonction affiche() que l'on aura créée
//on lui passe en paramètre la valeur du nombre de LED à éclairer
affiche(nombre_led);
}
void affiche(int valeur_recue)
{
//on éteint toutes les leds
digitalWrite(led_0, HIGH);
digitalWrite(led_1, HIGH);
digitalWrite(led_2, HIGH);
digitalWrite(led_3, HIGH);
//Puis on les allume une à une
if(valeur_recue >= 1)
{
digitalWrite(led_0, LOW);
}
if(valeur_recue >= 2)
{
digitalWrite(led_1, LOW);
}
if(valeur_recue >= 3)
{
digitalWrite(led_2, LOW);
}
if(valeur_recue >= 4)
{
digitalWrite(led_3, LOW);
}
}
```

6.7. TP 7 : Les entrées analogiques

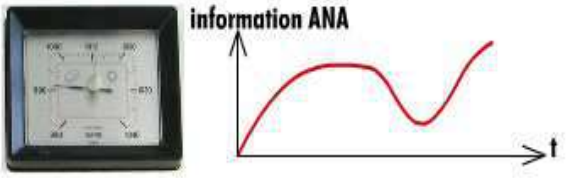
Mesurer une température, une pression, une tension, un débit, toutes ces grandeurs physiques qui sont des grandeurs analogiques. Comment faire ? Car le microprocesseur ne comprend que le binaire, il ne comprend que le numérique. Il va falloir convertir l'analogique en numérique...

☞ ***Ce qu'il faut savoir***

❖ ***Information analogique***

Elle caractérise l'état réal d'une grandeur physique. Elle peut prendre une infinité de valeurs. Elle varie continument.

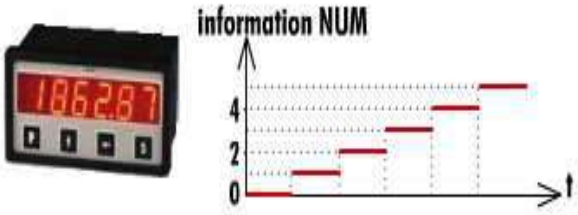
Exemple : température, vitesse d'un objet, pression, débit, ...



❖ ***Information numérique***

Elle caractérise une grandeur dénombrable, c'est-à-dire dont la variation d'un état à un autre s'effectue par incrément de manière discontinue.

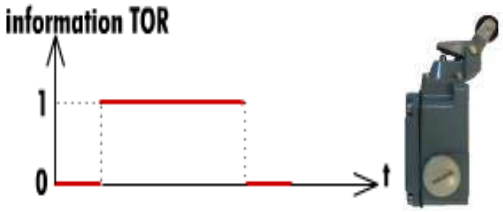
Exemple : nombre de tours réalisés par un moteur, quantité de pièces produites



❖ ***Information binaire (logique, TOR)***

Elle caractérise une grandeur qui ne peut prendre que 2 états : présent ou absent, vérifié ou non, vrai ou faux

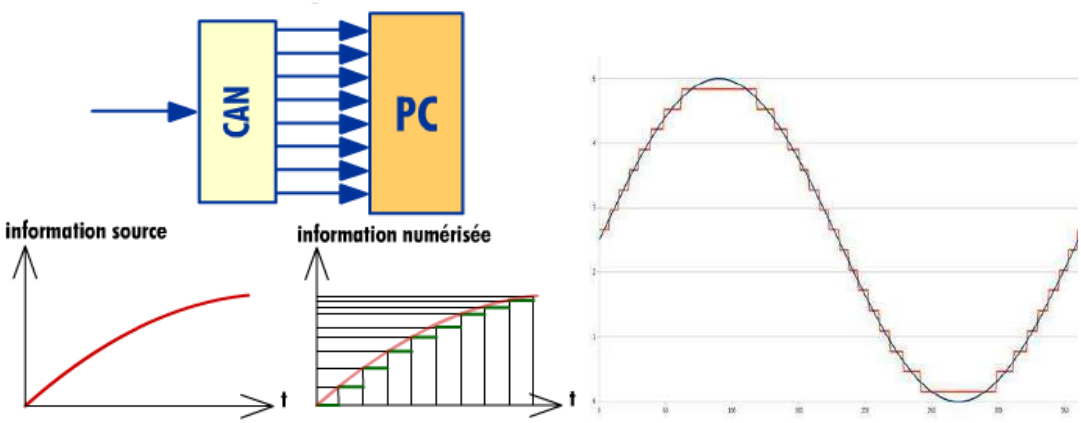
Exemple : tige de vérin sortie ou non, pièce présente ou non



☞ ***Convertisseur analogique numérique (CAN)***

En électronique numérique, on travaille avec des bits et des octets. En analogique, on travaille avec des grandeurs physiques : tension, courant, résistance, fréquence, etc.

Pour pouvoir exploiter des mesures analogiques avec un microcontrôleur, il faut convertir la mesure analogique (tension ou courant) en une grandeur numérique. C'est justement le but des convertisseurs analogique / numérique.



Sans entrer dans les détails techniques, certes très intéressants, mais aussi très compliqués, un convertisseur analogique / numérique permet de mesurer une tension (ou courant) (valeur analogique) et de représenter cette tension (ou courant) au moyen d'une valeur numérique.

L'idée est simple : associer une valeur numérique (un nombre entier) pour chaque valeur analogique d'une plage de tension (ou courant) bien précise.

☞ ***Analogique et Arduino***

Dans le cas d'une carte Arduino UNO, il y a 6 entrées analogiques, pouvant mesurer des tensions comprises entre 0 et 5 V, avec une précision de 10 bits (soit 1024 points = 2^{10}).

Si on fait rapidement le calcul, 1024 points sur une plage de 5V donne une précision absolue de 0,0048828125 V, soit environ 4,89mV.

☞ ***Précautions à prendre !***

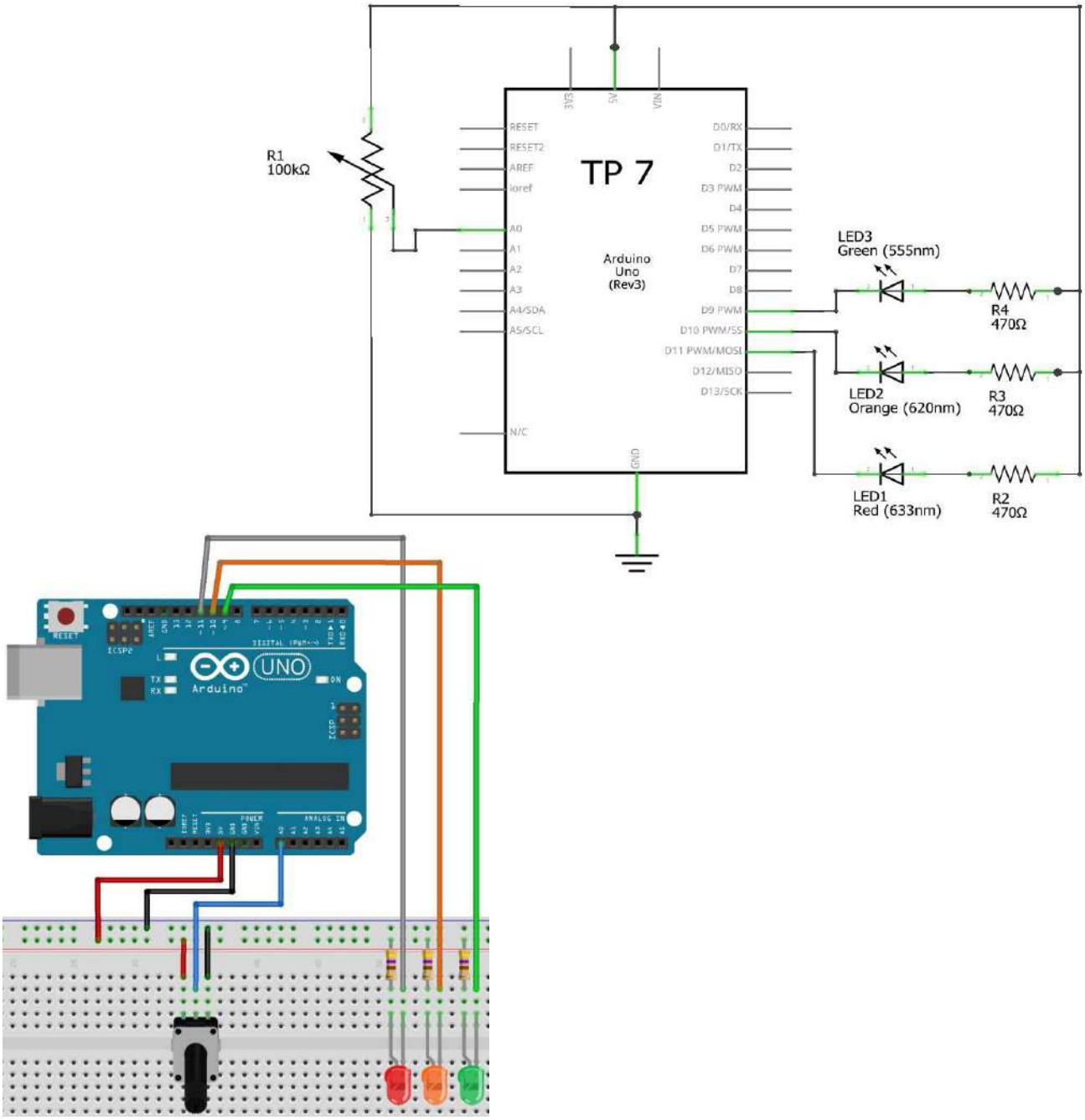
Avant de faire le montage du chapitre suivant, gardez en tête que :

- Injecter une tension supérieure à 5 volts ou inférieure à 0 volt sur une broche analogique endommagera immédiatement et définitivement votre carte Arduino.
- La précision de la mesure (10 bits) n'est pas modifiable,
- La mesure prend environ 100µs, cela fait un maximum de 10 000 mesures par seconde,
- Laisser une broche non connectée revient à avoir une antenne, mesurer une tension sur une broche non connectée retourne des valeurs de l'ordre de 300 à 500, même s'il n'y a pas de signal,
- Le résultat est sur 10 bits, soit entre 0 et 1023 (inclus).

☞ ***But :*** Manipuler les entrées analogiques et fonctions associées. Plusieurs variantes :

a. Afficher les valeurs du potentiomètre (en points et en Volts) via le moniteur série.
b. Idem mais en augmentant la précision (voir analogReference() pour plus d'infos)
c. Contrôler l'allumage de plusieurs LED en utilisant un potentiomètre. Suivant la position du potentiomètre, une certain nombre de LED seront allumées. Plus la valeur de sa résistance sera haute et plus le nombre de LED allumées sera grand.

☞ ***Schéma électrique :***



☞ ***Solution :***

- ***variante "a"***

```
int valPota; //Initialise la variable qui va recueillir la valeur du potentiomètre
void setup()
{
Serial.begin(9600); //Initialise la communication entre le PC et Arduino
}
void loop()
{
valPota = analogRead(A0); //Lis la valeur du potentiomètre
Serial.println(valPota); //Affiche la valeur du potentiomètre sur le moniteur série
  float tension = valPota * (5.0 / 1023.0);
  Serial.println(tension);

delay(200); //Attends 200 millisecondes
}
```

- ***variante "b"***

```
int valPota; //Initialise la variable qui va recueillir la valeur du potentiomètre
void setup()
{
Serial.begin(9600); //Initialise la communication entre le PC et Arduino
  // Améliore la précision de la mesure en réduisant la plage de mesure avec analogReference()

// DEFAULT : 5V
// EXTERNAL : Tension comprise entre 0 et 5 volts à injecter via la broche AREF.
// INTERNAL : 1,1 V
// INTERNAL1V1 : 1,1 V (Mega 2560)
// INTERNAL2V56 : 2,56V (Mega 2560)

  analogReference(INTERNAL);
  //analogReference(INTERNAL1V1); // Pour Arduino Mega2560
}
void loop()
{
valPota = analogRead(A0);
//Lis la valeur du potentiomètre
Serial.println(valPota); //Affiche la valeur du potentiomètre sur le moniteur série
  float tension = valPota * (1.1/ 1023.0);
  Serial.println(tension);

delay(200); //Attends 200 millisecondes
}
```

- ***variante "c"***

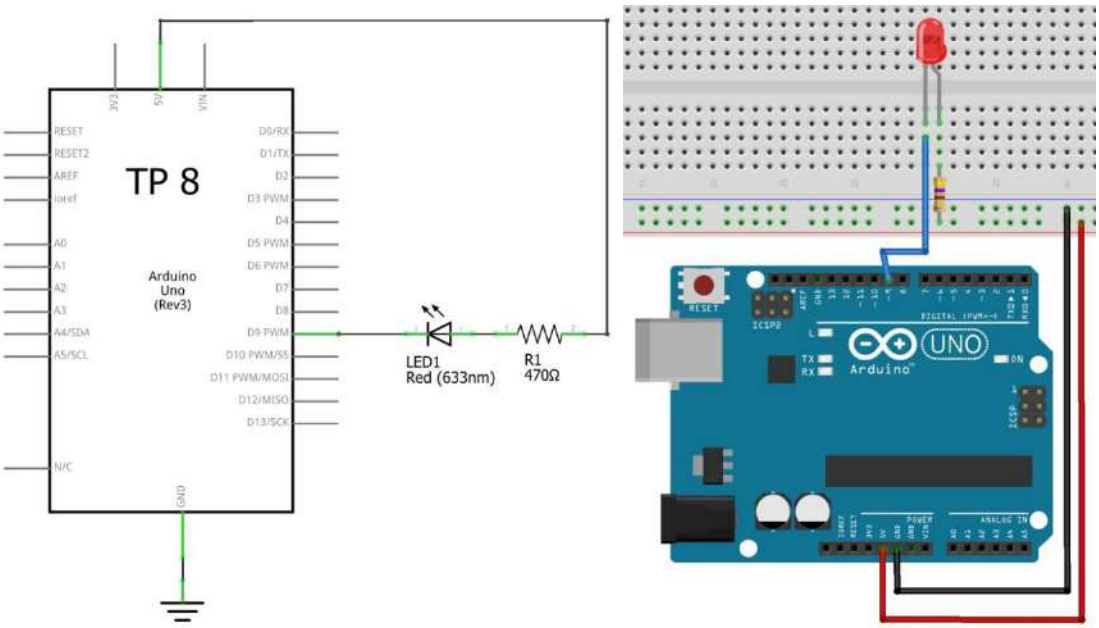
```
void setup()
{
pinMode(9, OUTPUT);
pinMode(10, OUTPUT);
pinMode(11, OUTPUT);
}
void loop()
{
int val = analogRead(0);
if (val > 750) {

digitalWrite(9, LOW);
digitalWrite(10, LOW);
digitalWrite(11, LOW);
} else if (val > 500) {

digitalWrite(9, LOW);
digitalWrite(10, LOW);
digitalWrite(11, HIGH);
} else if (val > 250) {

digitalWrite(9, LOW);
digitalWrite(10, HIGH);
digitalWrite(11, HIGH);
} else {

digitalWrite(9, HIGH);
digitalWrite(10, HIGH);
digitalWrite(11, HIGH);
}
delay(10);
}
```

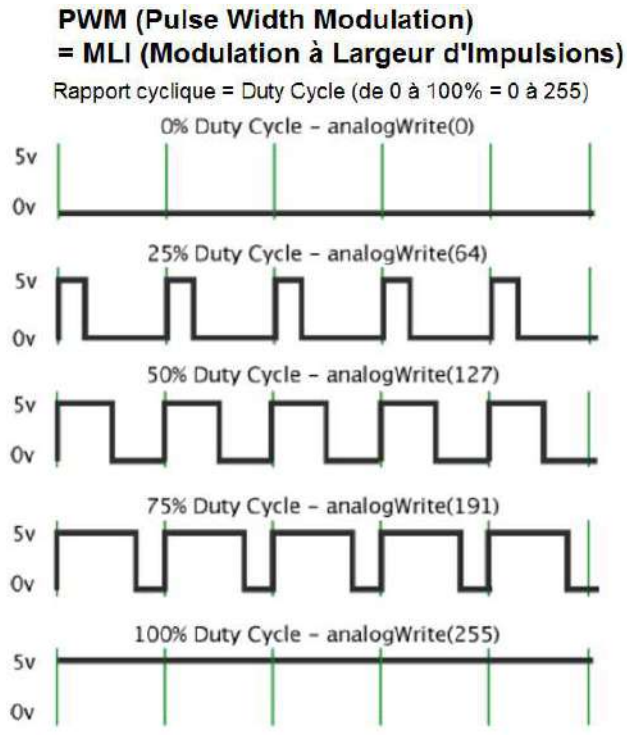
6.8. TP 8 : Les sorties analogiques (PWM)

☞ ***But :*** Manipuler les sorties analogiques (PWM) et fonctions associées. Plusieurs variantes :

a. Faire varier l'intensité d'une LED sans potentiomètre
b. Faire varier l'intensité d'une LED de manière ascendante et ensuite descendante
c. Faire varier l'intensité d'une LED suivant la position d'un potentiomètre (à rajouter)

☞ ***Schéma électrique :***





☞ ***Solution :***

- ***variante "a"***

```
void setup()
{
pinMode(9, OUTPUT);// La broche à laquelle la LED est connectée (pin 9 PWM )
}
void loop(){
for ( int val = 0 ; val < 255 ; val++ ) { // boucle incrémentant la variable val de 0 à 255, de 1 en 1
analogWrite(9, val); // Met la LED à sa nouvelle luminositée
delay(10);
}
}
```

- ***variante "b"1***

```
void setup()
{

pinMode(9, OUTPUT);// La broche à laquelle la LED est connectée (pin 9 PWM )

}
void loop(){
for ( int val = 0 ; val < 255 ; val++ ) { // boucle incrémentant la variable val de 0 à 255, de +1 en +1

analogWrite(9, val); // Met la LED à sa nouvelle luminositée

delay(10);
}
for ( int val = 255 ; val > 0 ; val-- ) { // boucle incrémentant la variable val de 0 à 255, de -1 en -1


analogWrite(9, val); // Met la LED à sa nouvelle luminositée

delay(10);
}
}
```

- ***variante "b"2***

```
int luminosite = 0;
int variation = 5;
void setup()
{
pinMode(9, OUTPUT);
}

void loop()
 {
analogWrite(9, luminosite);
luminosite = luminosite + variation;
if (luminosite == 0 || luminosite == 255)
 {
variation = -variation ;
 }
 delay(30);
}
```

- ***variante "c"***

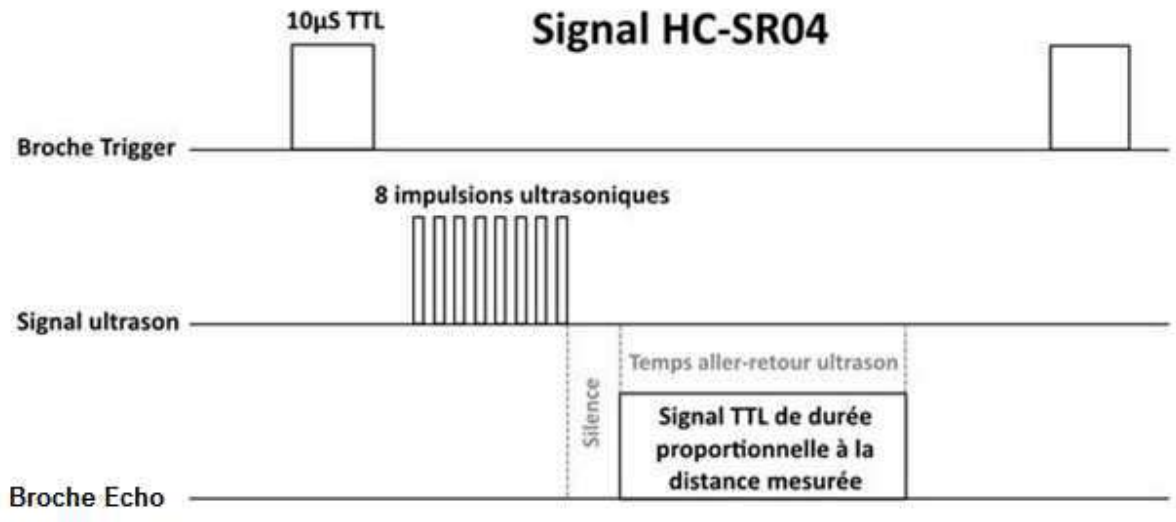
```
void setup()
{
pinMode(9, OUTPUT);// La broche à laquelle la LED est connectée (pin 9 PWM )
}
void loop()
{
int pot = analogRead(A0); // Lis la valeur du potentiomètre
int val = map(pot, 0, 1023, 0, 255); // on change d'intervalle
analogWrite(9, val); // Met la LED à sa nouvelle luminositée
delay(10);
}
```

6.9. TP 9 : Mesurer une distance

(Source : https://www.carnetdumaker.net/articles/mesurer-une-distance-avec-un-capteur-ultrason-hc-sr04-et-une-carte-arduino-genuino/)

☞ ***Ce qu'il faut savoir***

Le capteur HC-SR04 est un capteur à ultrason low cost. Ce capteur fonctionne avec une tension d'alimentation de 5 volts, dispose d'un angle de mesure de 15° environ et permet de faire des mesures de distance entre 2 centimètres et 4 mètres avec une précision de 3mm (en théorie, dans la pratique ce n'est pas tout à fait exact).



Le principe de fonctionnement du capteur est entièrement basé sur la vitesse du son.
Voilà comment se déroule une prise de mesure :

Pour déclencher une mesure, il faut présenter une impulsion "high" (5 V) d'au moins 10 µs sur la pin "Trig". Le capteur émet alors une série de 8 impulsions ultrasoniques à 40 kHz (inaudible pour l'être humain), puis il attend le signal réfléchi. Lorsque celui-ci est détecté, il envoie un signal "high" sur la sortie "Echo", dont la durée est proportionnelle à la distance mesurée.

☞ ***Quelques conseils d'utilisations :***

Ces capteurs à ultrason ont besoin d'une zone dégagée, avec une surface dure et lisse d'au moins 50cm² en face du capteur pour donner des résultats corrects.
Inutile d'essayer de mesurer une distance par rapport à un rideau ou une surface absorbant le son, ça ne donnera rien. De même, utiliser ce genre de capteur à ras le sol ou sur une table, voir pire, dans un tube ou une boite (déjà vu), ce n'est définitivement pas une bonne façon d'avoir des mesures correctes.

☞ ***Mesure de la distance de l'objet :***

La distance parcourue par un son dans l'air se calcule par la formule d'un MRU => e = V.t.

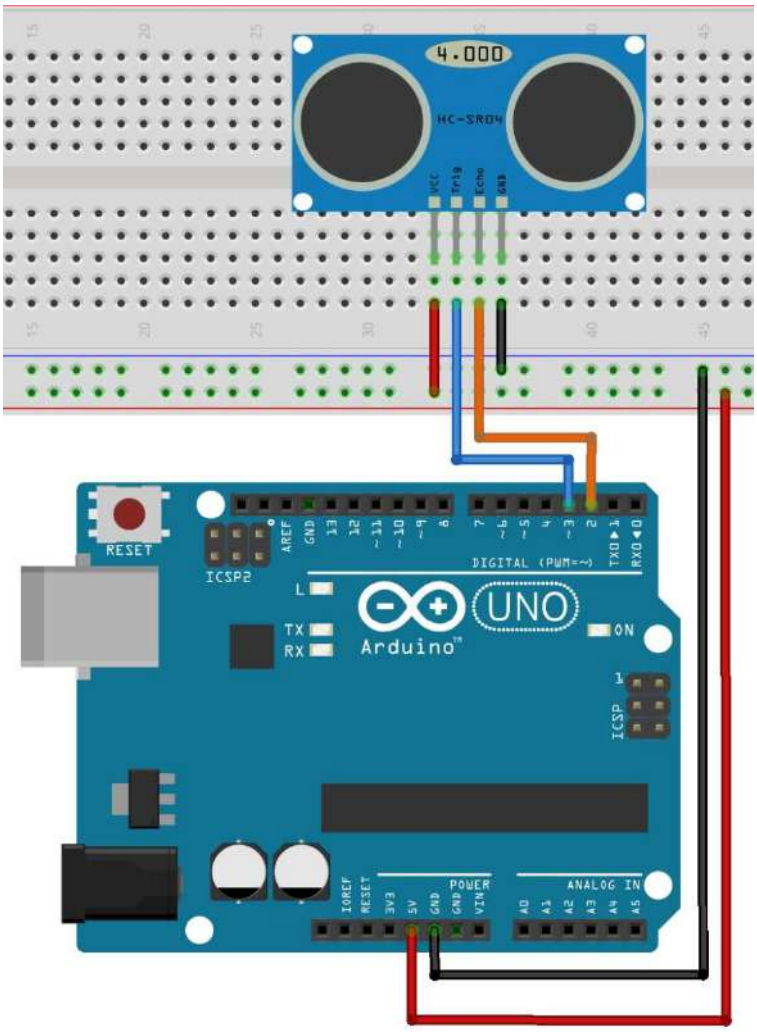
La vitesse du son dans l'air est de : $V = 340\,\frac{m}{s} = \frac{34\,000}{1\,000\,000}\,\frac{cm}{\mu s} = \frac{34}{1\,000}\,\frac{cm}{\mu s} = 0{,}034\ cm/\mu s$

Le capteur HC-SR04 donne une durée d'impulsions en µs. On sait aussi que le son fait un aller retour jusqu'au capteur. Il faut donc aussi en tenir compte.

En tenant compte de ces remarques, le calcul devient donc : $e = 0{,}034.\frac{t}{2}$

☞ ***But :*** Mesurer une distance avec le capteur à ultrasons HC-SR04 et l'afficher sur le moniteur série

☞ ***Schéma électrique :***



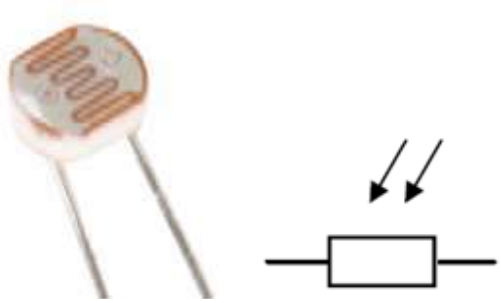
☞ ***Solution :***

```
//  Capteur à ultrasons (distance)
const int echoPin = 2;
const int triggerPin =3;
const unsigned long TIMEOUT = 25000UL; // 25 ms => 8m à 340 m/s
const float VITESSE_SON = 0.034; // V = 0,034 cm/µs
float distance = 0.0;


void setup()
{
Serial.begin(9600);
pinMode(echoPin, INPUT);
pinMode(triggerPin, OUTPUT);
digitalWrite(triggerPin, LOW);
}
void loop()
{
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);
    float Temps = pulseIn(echoPin, HIGH, TIMEOUT); // Temps fourni en us

        // N.B. La fonction pulseIn() retourne 0 si le temps de timeout est atteint. Il est donc possible de gérer l'absence d'obstacle si vous le souhaitez avec un if (Temps == 0) { ... } par exemple.

    // Distance = Vitesse du son * Temps => (attention, il faut diviser par 2 car aller retour)
    // Vitesse du so = 340 m/s = 34000 cm / 1 000 000 us = 0,034 cm/us

    distance = Temps* VITESSE_SON /2;

    Serial.print("La distance vaut : ");
    Serial.print(distance);
    Serial.println(" cm");
    delay(500);
}
```

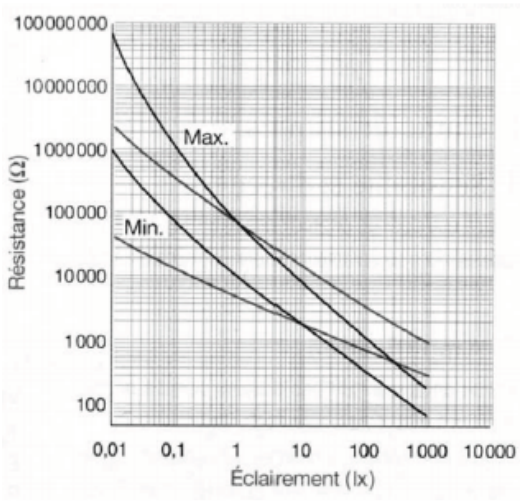
6.10. TP 10 : Mesurer la luminosité avec une LDR

☞ ***Petit rappel***

C'est une résistance variable, en fonction de la luminosité qu'elle reçoit. Sa résistance diminue quand elle reçoit de la lumière.



On s'en sert donc de capteur de luminosité. Non polarisée. Pour lire sa valeur avec une Arduino, il faut également l'associer avec une résistance équivalente à sa résistance maximum (dans le noir)



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	***Éclairement 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

☞ ***Remarque :***

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

☞ ***But :***

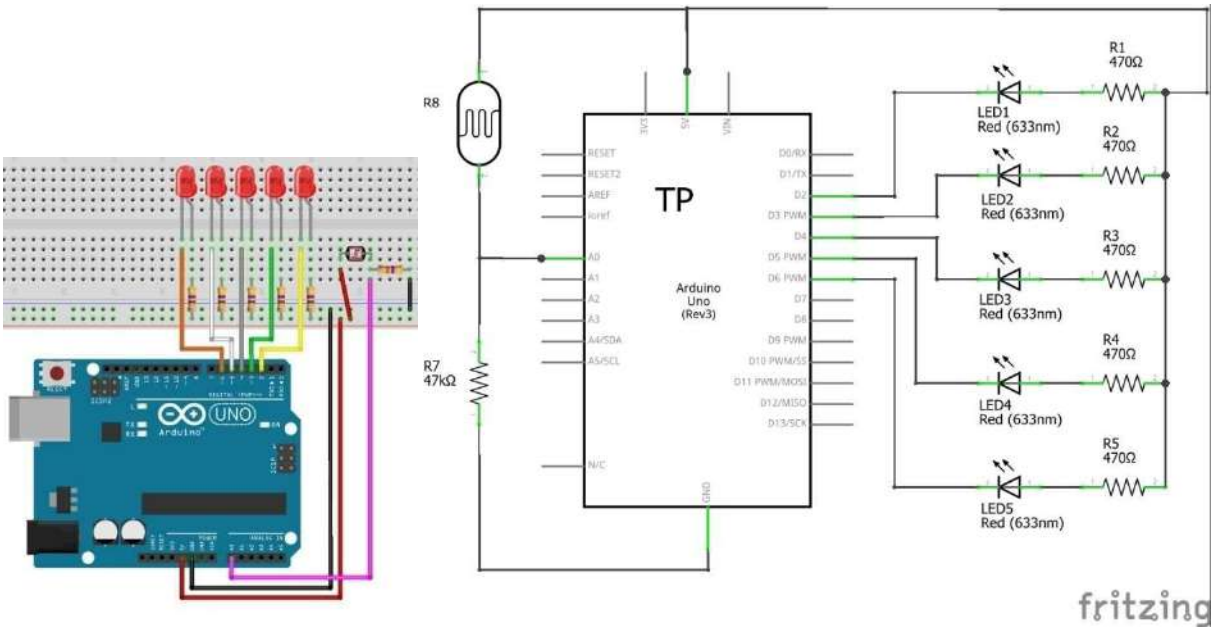
Mesurer la luminosité ambiante d'une pièce et envoyer la valeur mesurée via la liaison série. Cet exercice comporte plusieurs variantes :

a. Mesurer la luminosité ambiante et afficher la valeur lue sur le moniteur série
b. Mesurer la luminosité ambiante, la convertir en % et l'afficher sur le moniteur série
c. Mesurer la luminosité ambiante et allumer les 5 LEDs en fonction de cette mesure. Avec toujours affichage en % sur le moniteur série.

Que se passe-t-il si la LDR et la résistance R7 sont inversés ?

☞ ***Schéma électrique :***

La résistance R7 sera peut-être à adapter en fonction de la LDR de l'atelier !



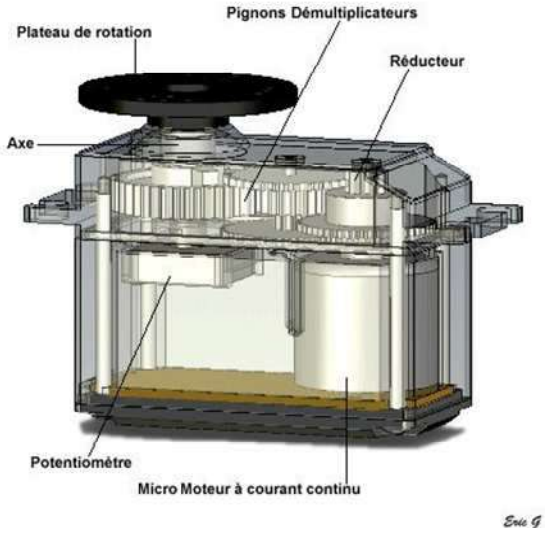
☞ ***Solution :***

à faire !

6.11. TP 11 : Le servomoteur

☞ ***Ce qu'il faut savoir***

Un servomoteur est un type de moteur électrique. C'est un dispositif typiquement utilisé en modélisme pour, par exemple, contrôler la direction d'une voiture télécommandée.



Sur un servomoteur, l'angle de l'axe reste fixé dans une position et peu varier entre 0 et 180° en fonction du signal envoyé. Un servomoteur comprend :

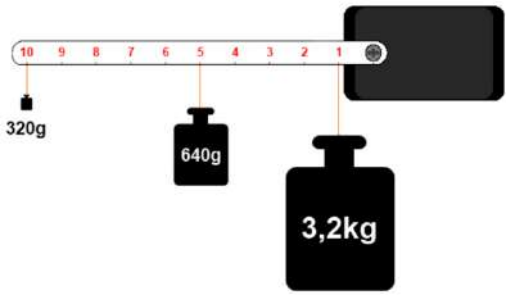
- **Un moteur électrique continu**
- **Des engrenages** réducteur en sortie de ce moteur (pour avoir moins de vitesse et plus de couple).
- **Un capteur** type "potentiomètre" raccordé sur la sortie. Ce qui permet de mesurer l'angle de rotation sur l'axe de sortie.
- **Un asservissement** électronique pour contrôler la position/rotation de cet axe de sortie

☞ ***Et le couple ?***

Le réducteur fait deux choses : d'une part il réduit la vitesse de rotation en sortie de l'axe du servomoteur (et non du moteur CC), d'autre part il permet d'augmenter le couple en sortie du servomoteur (là encore non en sortie du moteur CC).

Les moteurs CC se débrouillent très bien pour tourner très vite mais lorsqu'ils font une si petite taille ils sont bien moins bons pour fournir du couple. On va donc utiliser ce réducteur qui va réduire la vitesse, car nous n'avons pas besoin d'avoir une vitesse trop élevée, et augmenter le couple pour ainsi pouvoir déplacer une charge plus lourde.

$$C = F.d$$

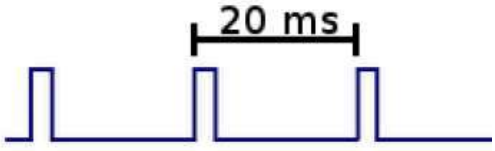


☞ ***Commander un servomoteur ?***

La consigne envoyée au servomoteur n'est autre qu'un signal électronique de type PWM. Il dispose cependant de deux caractéristiques indispensables pour que le servomoteur puisse comprendre ce qu'on lui demande. À savoir : une fréquence fixe de valeur 50Hz et d'une durée d'état HAUT elle aussi fixée à certaines limites.

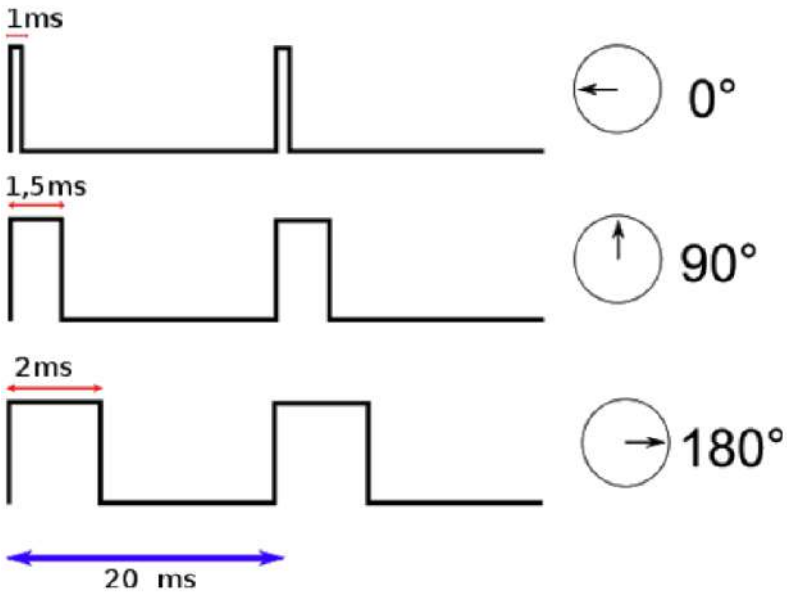
Certains sites de modélisme font état d'un nom pour ce signal : une PPM pour *Pulse Position Modulation* Le signal que nous allons devoir générer doit avoir une fréquence de 50 Hz. Autrement dit, le temps séparant deux fronts montants est de 20 ms.

Malheureusement, la fonction analogWrite() de Arduino ne possède pas une fréquence de 50Hz, mais de 500Hz environ. On ne pourra donc pas utiliser cette fonction.



Il va donc falloir générer un signal de 50 Hz et avec un état HAUT d'une durée modifiable. Et c'est cette durée de l'état HAUT qui fixera la position du bras du servomoteur à un angle déterminé. En résumé, un signal ayant une durée d'état HAUT très faible donnera un angle à 0°, le même signal avec une durée d'état HAUT plus grande donnera un angle au maximum de ce que peut admettre le servomoteur.

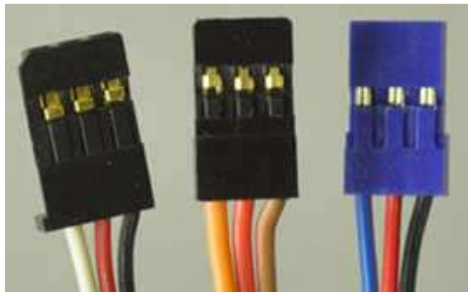
Le temps d'impulsion varie d'un fabricant à l'autre, mais les valeurs suivantes sont assez standards



☞ ***Raccordement d'un servomoteur***

Les servomoteurs sont pilotés par un fil de commande et alimentés par deux autres fils. Habituellement, ces 3 fils sont rassemblés dans une prise au format standard.

- Brun ou noir = MASSE/GND (borne négative de l'alimentation)
- Rouge = alimentation servo (Vservo, borne positive de l'alimentation) 4,5 à 6V en général
- Orange, jaune, blanc ou bleu = Commande de position du servo



☞ ***Connaissances nécessaires au pilotage d'un servomoteur***

Pour utiliser le servo avec Arduino, il va nous falloir générer le signal PPM vu précédemment. C'est à dire créer un signal d'une fréquence de 50Hz et modifier l'état haut d'une durée comprise entre 1 et 2ms. Contraignant n'est-ce pas ? C'est pourquoi l'équipe d'Arduino a été sympa en implémentant une librairie très bien nommée : "Servo". Tout comme l'objet Serial vous permettait de faire abstraction du protocole de la voie série, l'objet Servo va vous permettre d'utiliser les servomoteurs

- ***#include <Servo.h>*** : L'utilisation de la librairie servo.h qui prend en charge la communication entre l'Arduino et le servomoteur.
- ***Servo.monservo*** : La création d'un objet de type Servo, appelé ici "monservo"
- ***monservo.attach (2)*** : Le fil de commande ce servo sera connecté au PIN 2
  - monservo.attach(2, 1000, 2000); // complet avec ses arguments (optionnel !)

Cette fonction prend 3 arguments :

- Le numéro de la broche sur laquelle est relié le fil de signal
- La valeur basse (angle à 0°) de la durée de l'état haut du signal de PPM en microsecondes (optionnel, défaut à 544 µs)
- La valeur haute (angle à 90°, 180°, 360°, etc.) de la durée de l'état haut du signal de PPM en microsecondes (optionnel, défaut à 2400 µs)

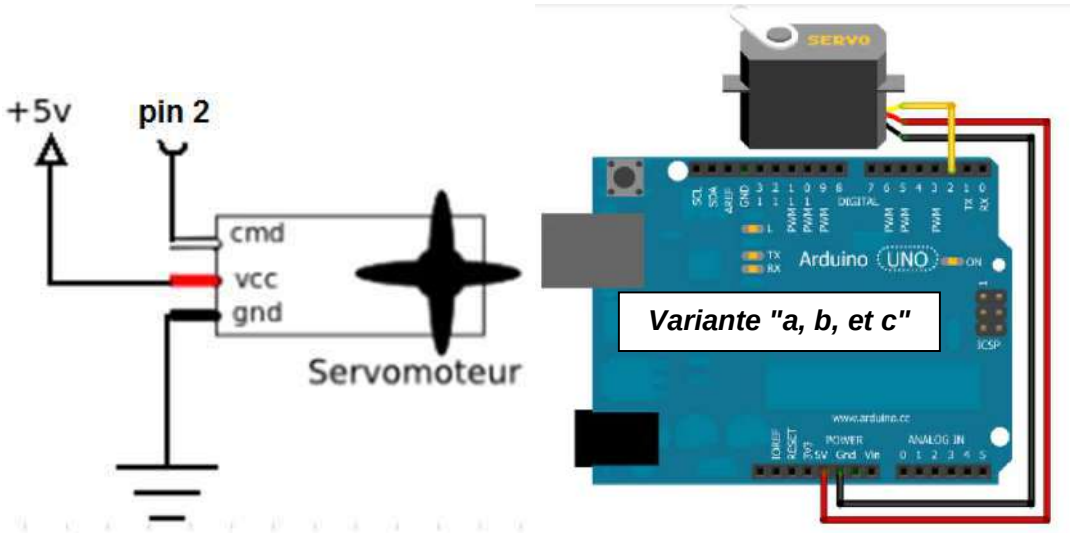
- ***monservo.write (Angle)*** : Demander au servomoteur de se déplacer à l'angle désiré.
- ***monservo.read (Angle)*** : Pour lire la valeur de l'angle du servomoteur.

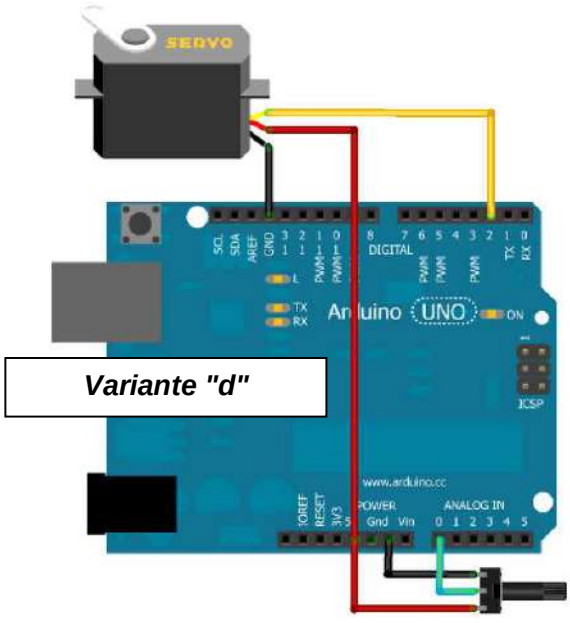
☞ ***But :***

Manipuler un servomoteur et ses fonctionnalités. Cet exercice comporte plusieurs variantes :

a. Positionner le servomoteur à 0°
b. Positionner le servomoteur à 180°
c. Effectuer un balayage entre 0° et 180° puis entr e 180° et 0° avec un certain délai entre chaque position
d. Le servomoteur doit “suivre” le potentiomètre. (En tournant l’axe du potentiomètre, le bras du servomoteur doit tourner à son tour et dans le même sens). La valeur du potentiomètre sera affichée via la liaison série

☞ ***Schéma électrique :***





☞ ***Solution :***

- ***variante "a et b"***

```
// Test utilisation servomoteur SG90 //
#include <Servo.h>
Servo monservo;  // crée l'objet pour contrôler le servomoteur
void setup()
{
 monservo.attach(2);  // utilise la broche 9 pour le contrôle du servomoteur
 monservo.write(0); // variante a : positionne le servomoteur à 0°
 monservo.write(180); // variante b : positionne le servomoteur à 180°
}
void loop()
{
}
```

- ***variante "c"***

```
#include <Servo.h>
Servo myservo;
int pos = 0;
void setup()
{
 myservo.attach(9);
}
void loop()
{
 for(pos = 0; pos < 180; pos += 1)
 {
  myservo.write(pos);
  delay(15);
 }
 for(pos = 180; pos>=1; pos-=1)
 {
  myservo.write(pos);
  delay(15);
 }
}
```

- ***variante "d"***

```
#include //On oublie pas d'ajouter la bibliothèque !
const int potar = 0; //notre potentiomètre
Servo monServo;
void setup()
{
//on déclare le servo sur la broche 2 (éventuellement régler les bornes)
monServo.attach(2);
//on oublie pas de démarrer la voie série
Serial.begin(9600);
}
void loop()
{
//on lit la valeur du potentiomètre
int val = analogRead(potar);
//mise à l'échelle de la valeur lue vers la plage [0;180]
int angle = map(val, 0, 1023, 0, 180);
//on met à jour l'angle sur le servo
monServo.write(angle);
//on renvoie l'angle par la voie série pour superviser
Serial.println(angle);
//un petit temps de pause
delay(100);
}
```

6.12. TP 12 : Mesurer une température positive

☞ ***Pré requis :*** Il est nécessaire de bien comprendre les entrées analogiques pour réaliser cet atelier pratique

☞ ***Ce qu'il faut savoir***

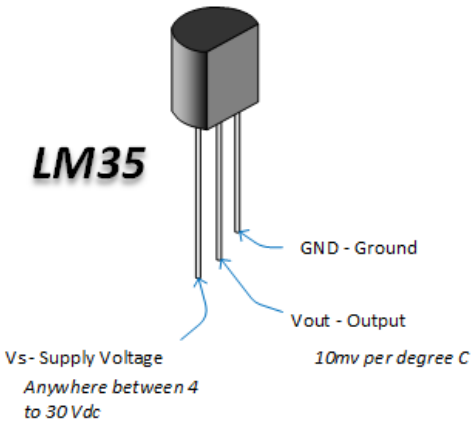
Le capteur de température LM35 est un capteur analogique de température fabriqué par Texas Instruments. Il est extrêmement populaire en électronique, car précis, peu couteux, très simple d'utilisation et d'une fiabilité à toute épreuve.

Il existe plusieurs variantes du capteur de température LM35 :

- LM35DZ : de 0 à 100℃ (précision de 1.5℃)
- LM35CZ : de -40 à 110℃ (précision de 1.5℃)
- LM35CAZ : de -40 à 110℃ (précision de 1℃)

La tension de sortie du capteur de température LM35 est une fonction linéaire de la température.

Comme l'explique la formule suivante, pour 0℃, la tension de sortie vaut 0V. La tension de sortie augmente de 10mV à chaque augmentation de 1℃.



$$V_{outLM35} = \left(10\frac{mV}{°C}\right).T(°C)$$

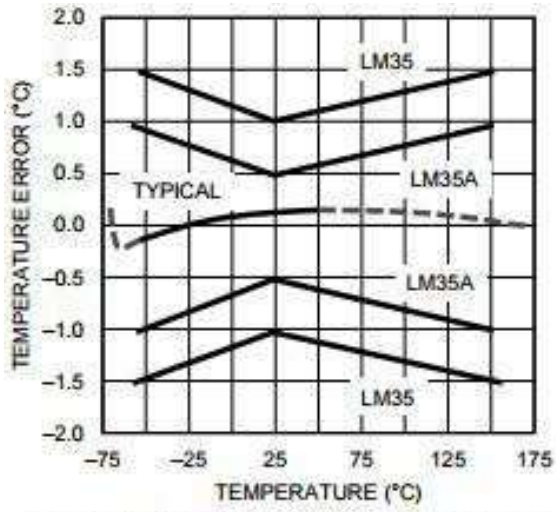
Une des grandes forces du capteur LM35, qui fait sa popularité, c'est sa pré-calibration en sortie d'usine. Tous les capteurs LM35 sont calibrés en degré Celsius lors de la fabrication. Cela signifie que vous n'avez absolument rien à faire pour calibrer le capteur, il l'est déjà au moment où vous le sortez de son sachet !

La précision garantie par le fabricant est de :

- **LM35 :** +/-1℃ à 25℃ et +/-1.5℃ à -55℃ et +150℃
- **LM35A :** +/-0.5℃ à 25℃ et +/-1℃ à -55℃ et +150℃.

Une autre des grandes forces du capteur LM35, c'est sa linéarité : moins de 1℃ d'erreur sur la plage complète de -55℃ à +150℃.

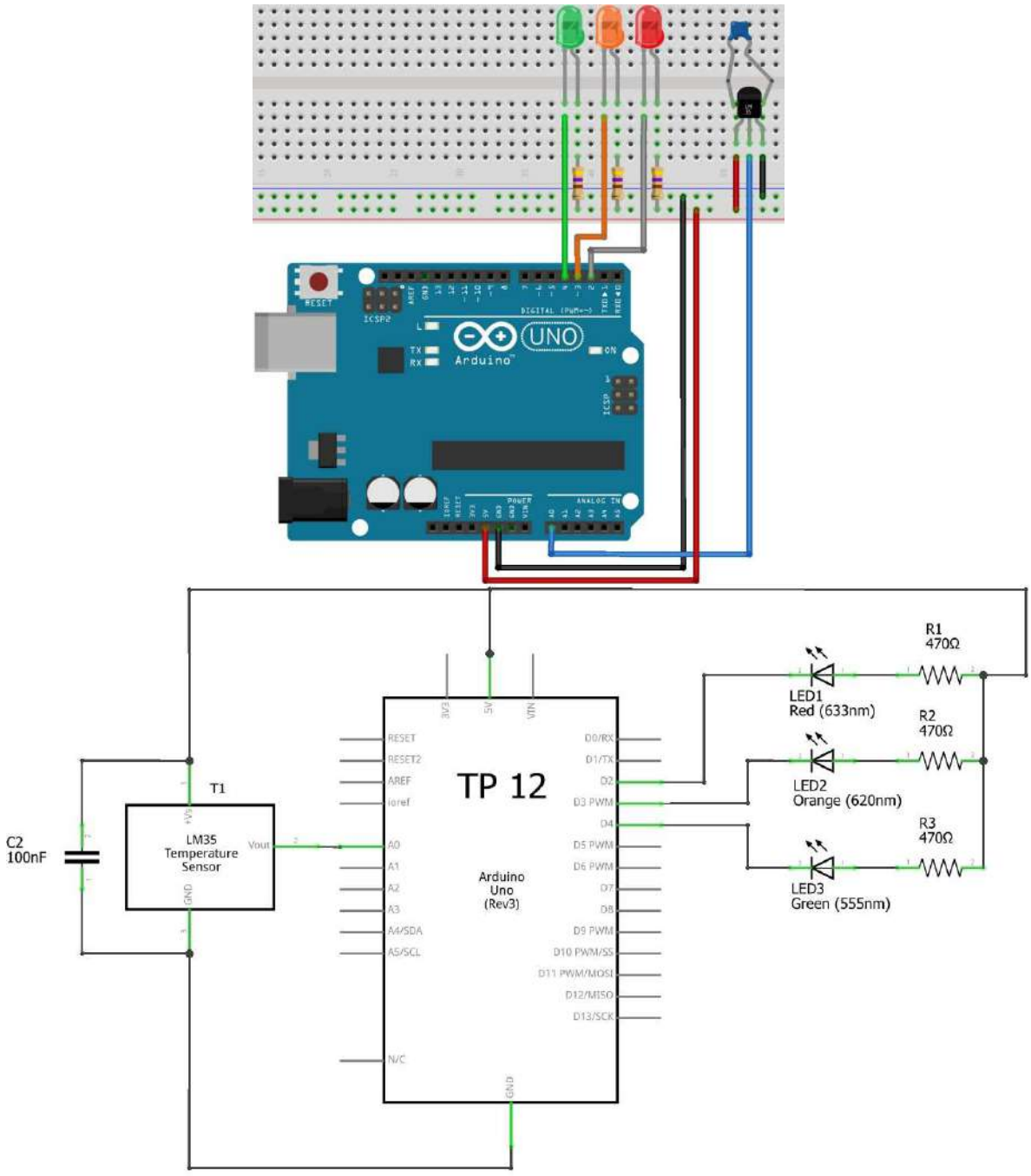
Comme on a 10mv/℃ et que la sortie du capteur est (quasi) parfaitement linéaire, convertir une mesure en température se résume à faire un simple produit en croix.



☞ ***But :*** Manipuler un capteur de température LM35. Cet exercice comporte plusieurs variantes :

a. Mesurer la température ambiante de l'atelier et de la transformer en degré Celsius pour ensuite l'afficher via le moniteur série.
b. Idem mais en augmentant la précision (voir analogReference() pour plus d'infos)
c. Idem avec allumage LED rouge si T>=30℃, orange si 25℃ < T < 30 ℃ et verte si T <= 25℃

☞ ***Schéma électrique :***



☞ ***Solution :***

- ***variante "a"***

```
// Code d'exemple pour le capteur LM35 (2℃ ~ +110℃).
voidsetup() {
// Initialise la communication avec le PC
  Serial.begin(9600);
}

voidloop() {

// 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);
}
```

- ***variante "b"***

```
/*
 * Code d'exemple pour le capteur LM35 (2℃ ~ +110℃) avec une meilleure précision.
 */

void setup()
{
  // Initialise la communication avec le PC
  Serial.begin(9600);

  // Améliore la précision de la mesure en réduisant la plage de mesure
  analogReference(INTERNAL); // Pour Arduino UNO
  // analogReference(INTERNAL1V1); // Pour Arduino Mega2560
}

void loop() {

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

  // Transforme la mesure (nombre entier) en température via un produit en croix
  float temperature_celcius = valeur_brute * (1.1 / 1023.0 * 100.0);

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



// Allumage LED à faire !!
```

6.13. TP 13 : Mesurer une température négative

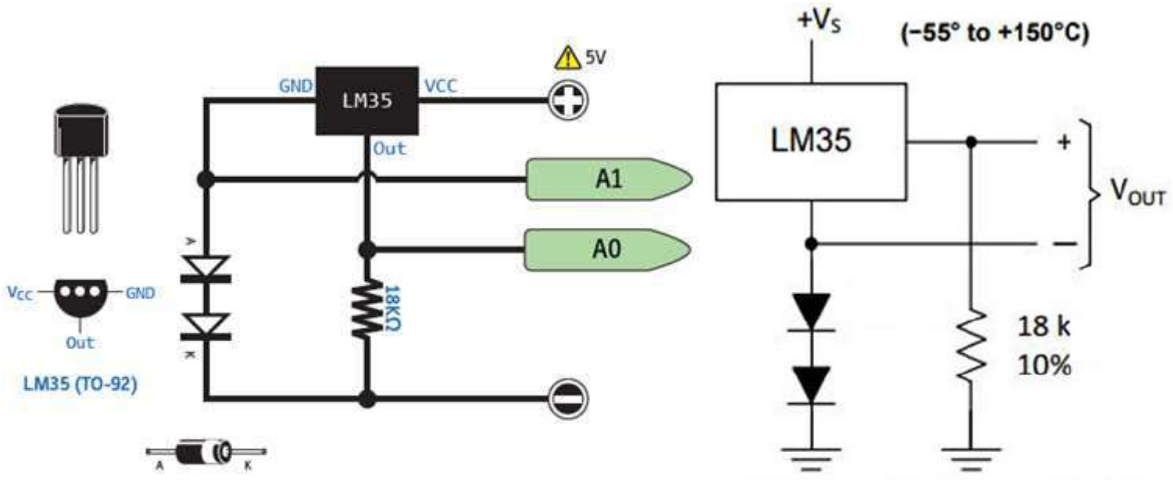
Mesurer des températures comprises entre 0℃ et +11 0℃, c'est bien. Cependant, il arrive régulièrement qu'on ait besoin de mesurer des températures négatives. Pour ce faire, il nous faut un capteur LM35 CZ ou CAZ (attention, pas un DZ !),

☞ ***Comment ça marche ?***

Les deux diodes induisent une chute de tension d'environ 0,6V à leur borne chacune, soit 1.2V au total entre la masse réelle du montage et la broche `GND` du capteur. Avec une masse à 1,2V au niveau du capteur, il est possible pour le capteur d'aller en dessous de "son" 0V vu qu'il n'est en réalité pas à 0V, mais à 1.2V.

Il y a cependant un petit souci avec ce montage : la tension aux bornes des diodes varie en fonction de la température. Ce qui est embêtant pour un montage censé mesurer des températures. C'est un comportement normal pour une diode, sauf que dans notre cas, c'est un comportement parasite.

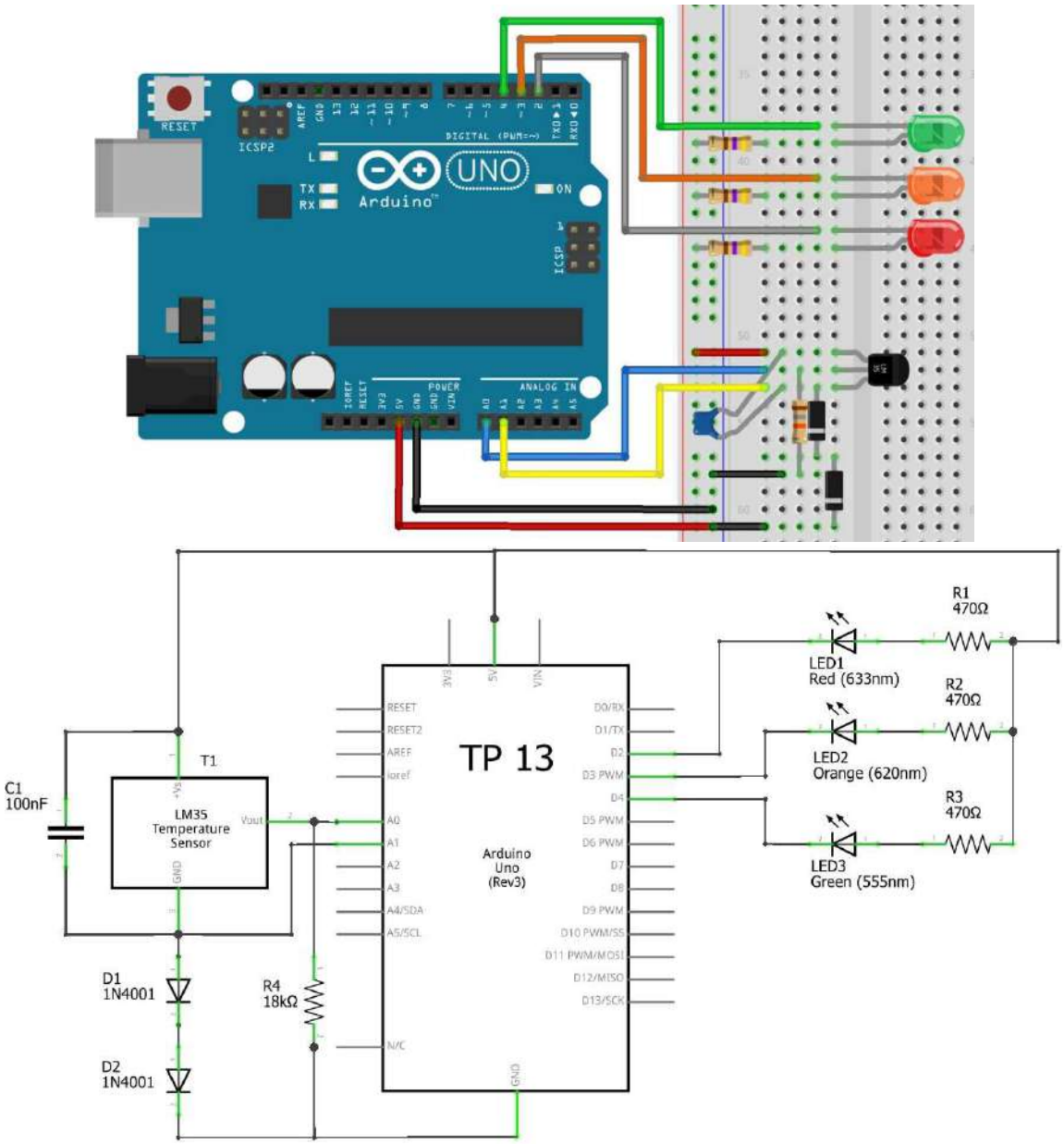
L'astuce pour "annuler" cette variation de tension en fonction de la température consiste à mesurer la tension au niveau de la broche `GND` du capteur, puis de corriger la mesure finale en faisant une soustraction dans le code.



☞ ***But :*** Manipuler un capteur de température LM35. Cet exercice comporte plusieurs variantes :

a. Mesurer la température ambiante de l'atelier et de la transformer en degré Celsius pour ensuite l'afficher via le moniteur série.
b. Idem avec allumage LED rouge si T>=10℃, orange si 0℃ < T < 10 ℃ et verte si T <= 0℃

☞ ***Schéma électrique :***



☞ ***Solution :***

- ***variante "a"***

```
/*
 * Code d'exemple pour le capteur LM35 avec support des températures négatives (-40°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 tant que la carte Arduino est alimentée
void loop() {

  // Mesure la tension sur la broche A0 (sortie capteur) et A1 (référence du point zéro)
  int valeur_brute = analogRead(A0);
  int valeur_offset = analogRead(A1);

  // Transforme la mesure (nombre entier) en température via un produit en croix
  float temperature_celcius = (valeur_brute - valeur_offset) * (5.0 / 1023.0 * 100.0);

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

- ***variante "b"***

// Allumage LED à faire !!

6.14. TP 13 : Le moteur pas à pas

☞ ***But :***

☞ ***Schéma électrique :***

☞ ***Solution :***

6.15. TP 14 : Le moteur à courant continu

☞ ***But :***

☞ ***Schéma électrique :***

☞ ***Solution :***

6.16. TP 15 : Afficheur 7 segments

https://www.carnetdumaker.net/articles/utiliser-un-afficheur-7-segments-avec-une-carte-arduino-genuino/

https://www.carnetdumaker.net/articles/faire-une-barre-de-progression-avec-arduino-et-liquidcrystal/

- ***But :***

- ***Schéma électrique :***

- ***Solution :***

https://www.carnetdumaker.net/articles/la-gestion-du-temps-avec-arduino/