Les codes utilisés dans ce cours peuvent être téléchargés à l'adresse suivante:

https://arduino.education/codes/codes.zip

Ce document est une compilation et une adaptation de textes et d'exercices, depuis les sources suivantes:

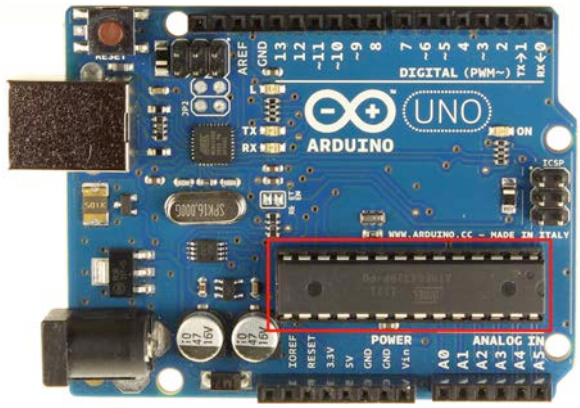
Sources principales:

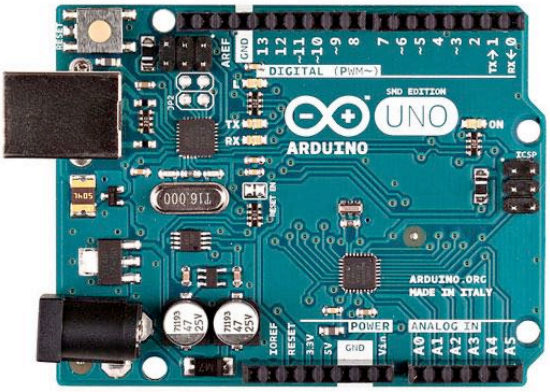
- http://arduino.cc/fr/
- http://eskimon.fr/
- http://eskimon.fr/ebook-tutoriel-arduino
- https://zestedesavoir.com/tutoriels/686/arduino-premiers-pas-en-informatique-embarquee/
- http://mediawiki.e-apprendre.net/index.php/Diduino-Robot
- https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino
- https://www.didel.com

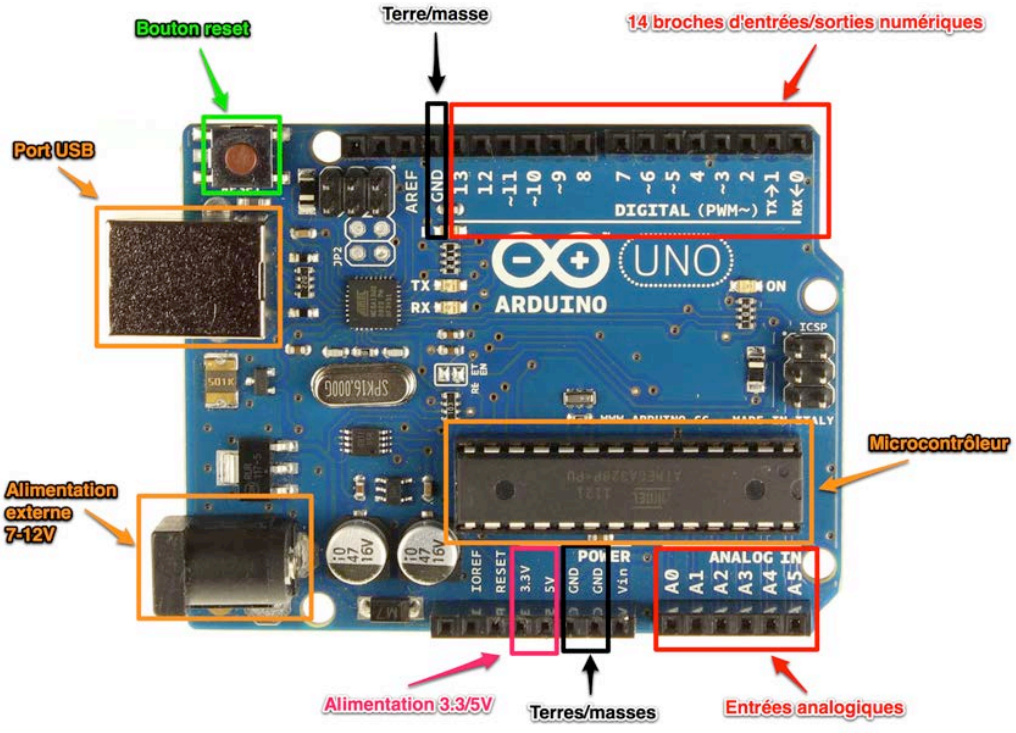
Sources annexes:

- http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ARDUINO
- http://chamayou.franck.free.fr/spip/spip.php?article177
- http://makezine.com/category/technology/arduino/
- http://www.craslab.org/arduino/livrethtml/LivretArduinoCRAS.html
- http://arduino103.blogspot.ch
- http://www.semageek.com

Arduino Uno	Arduino Uno SMD







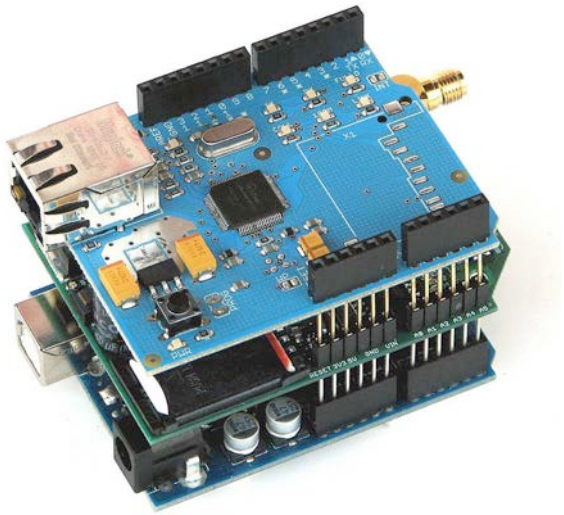
L'alimentation

Pour fonctionner, une carte Arduino a besoin d'une alimentation. Le microcontrôleur fonctionnant sous 5V, la carte peut être alimentée directement en 5V par le port USB ou bien par une alimentation externe qui est comprise entre 7V et 12V. Un régulateur se charge ensuite de réduire la tension à 5V pour le bon fonctionnement de la carte.

Attention: les cartes **Arduino Due ainsi que d'autres cartes récentes** fonctionnent avec un voltage de 3.3V au niveau des sorties! Le voltage de l'alimentation est similaire à l'Arduino Uno. Dans ce cours, nous partons du principe que le voltage des montages est en 5 Volts.

La connectique

A part une LED sur la broche 13, la carte Arduino ne possède pas de composants (résistances, diodes, moteurs...) qui peuvent être utilisés pour un programme. Il est nécessaire de les rajouter. Mais pour cela, il faut les connecter à la carte. C'est là qu'interviennent les connecteurs, aussi appelés **broches** (*pins*, en anglais).

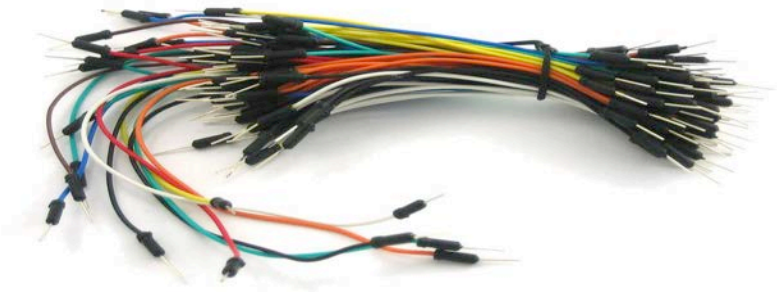


Sur les Arduino et sur beaucoup de cartes compatibles Arduino, les broches se trouvent au même endroit. Cela permet de fixer des cartes d'extension, appelée *shields* en les empilant.

Exploration des broches Arduino

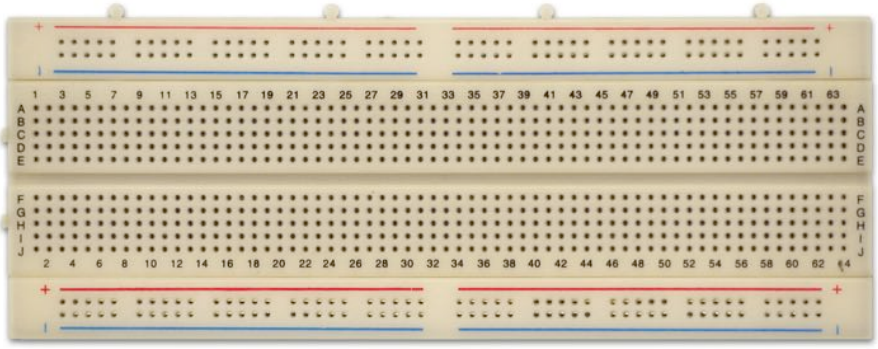
- **0 à 13** Entrées/sorties numériques
- **A0 à A5** Entrées/sorties analogiques
- **GND** Terre ou masse (0V)
- **5V** Alimentation +5V
- **3.3V** Alimentation +3.3V
- **Vin** Alimentation non stabilisée (= le même voltage que celui à l'entrée de la carte)

Les connexions entre les composants sont réalisées par des *jumpers*, sortes de petits câbles.

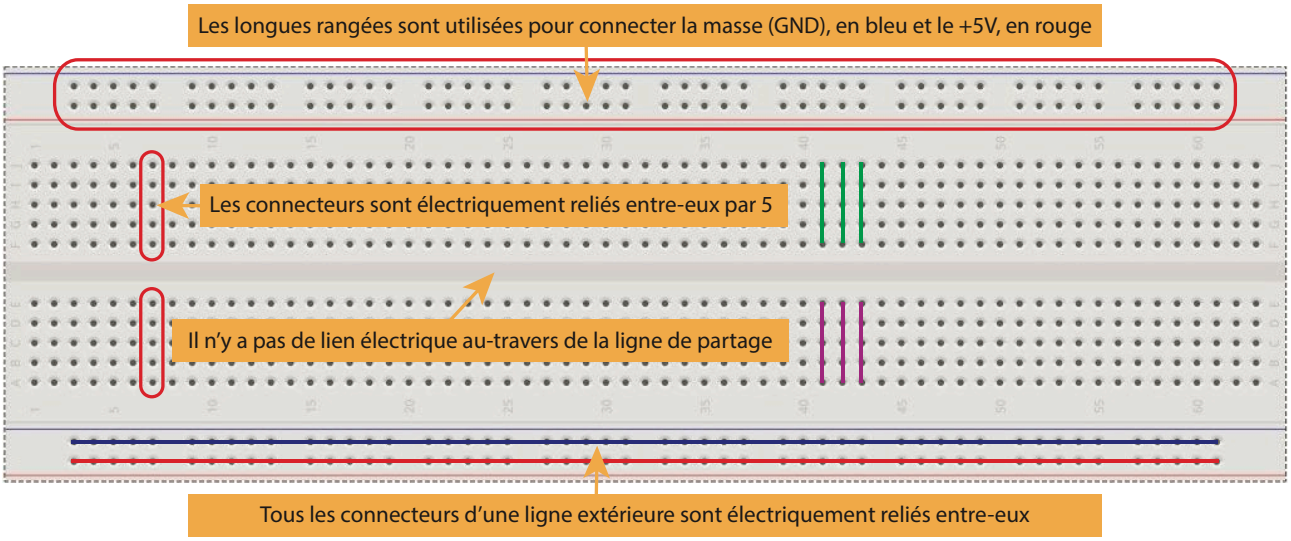


La platine d'expérimentation (breadboard)

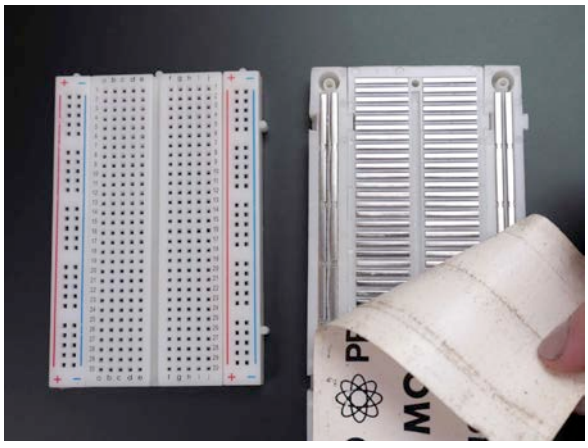
Une platine d'expérimentation (appelée *breadboard*) permet de réaliser des prototypes de montages électroniques sans soudure et donc de pouvoir réutiliser les composants.

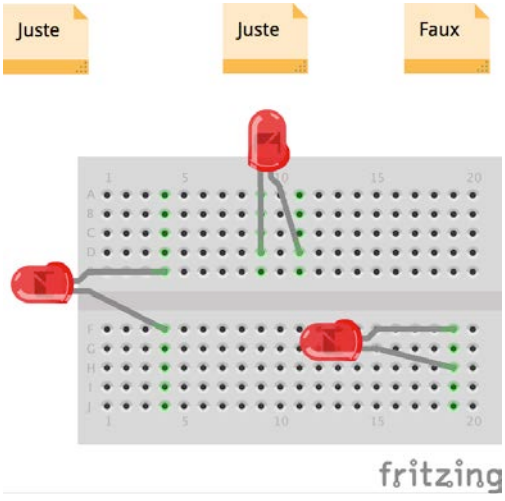


Tous les connecteurs dans une rangée de 5 sont reliés entre eux. Donc si on branche deux éléments dans un groupe de cinq connecteurs, ils seront reliés entre eux. Il en est de même des alignements de connecteurs rouges (pour l'alimentation) et bleus (pour la terre). Ainsi, les liens peuvent être schématisés ainsi:



Les composants doivent ainsi être placés à cheval sur des connecteurs qui n'ont pas de liens électriques entre eux, comme sur le schéma ci-contre.

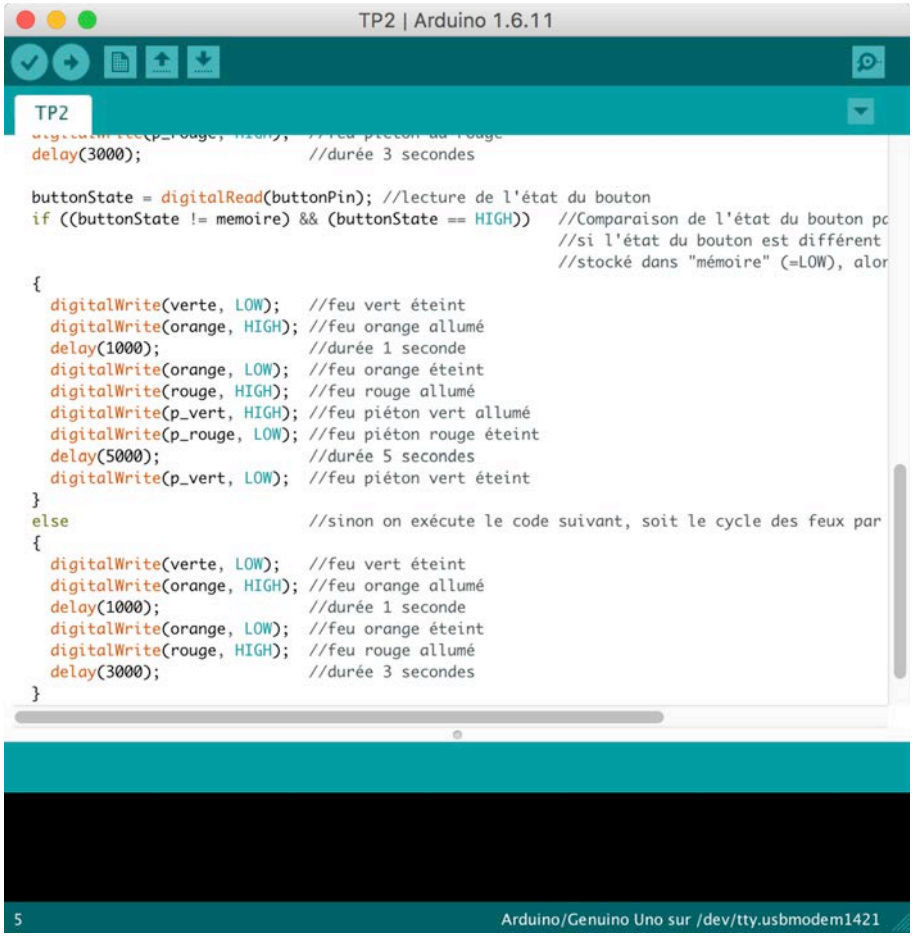




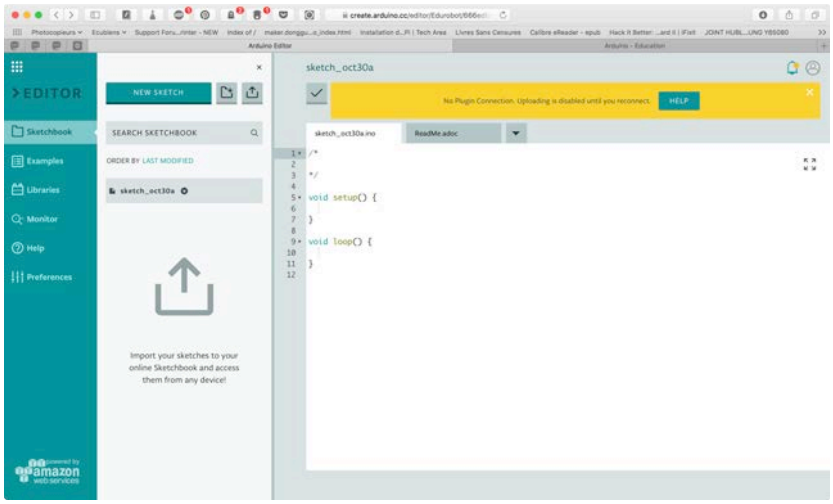
Le logiciel Arduino IDE

Le logiciel Arduino IDE fonctionne sur Mac, Windows et Linux. C'est grâce à ce logiciel que nous allons créer, tester et envoyer les programmes sur l'Arduino.

L'IDE est téléchargeable à l'adresse suivante: http://arduino.cc.



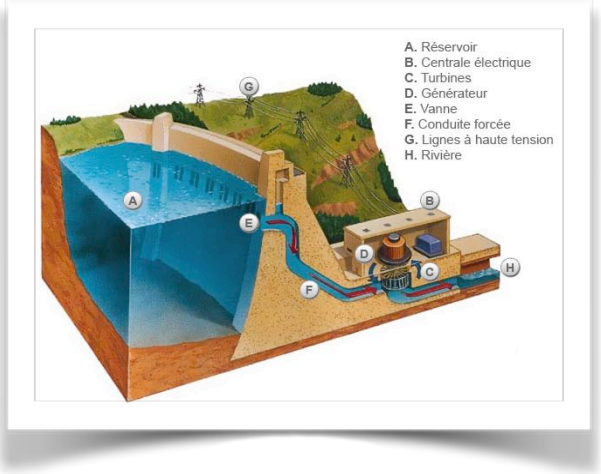
Une version en ligne de l'éditeur de code est disponible à cette adresse: https://create.arduino.cc/editor



Les bases de l’électronique

Petit rappel sur l’électricité

L’électricité (ou plus précisément *courant électrique*) est un déplacement d’électrons dans un milieu conducteur. On peut se l'imaginer comme un déplacement d'eau dans un tuyau ou à un barrage hydroélectrique.



- Les électrons seraient l’eau
- Le générateur serait le réservoir d’eau
- Les conducteurs sont naturellement les conduites forcées
- Le consommateur (une ampoule ou une diode, par exemple) est la turbine, qui exploite l’énergie du déplacement des électrons

Sur un barrage, plus la différence entre l’altitude du niveau du réservoir et celui de la turbine est importante, plus la pression de l’eau sera importante. Pour un barrage on appelle cette différence d’altitude *hauteur de chute*. Cela équivaut sur un circuit électrique à la *différence de potentiel*, qui se mesure en *Volts* (V) et se note *U.*
Le *débit de l’eau* (=la quantité d’eau qui s’écoule par unité de temps) correspond à *l’intensité*, aussi appelée *courant*, qui est donc le débit d’électrons. Elle se mesure en *Ampères* (A) et se note *I.*
La puissance électrique se note *P* et se mesure en *Watts* (W). Elle exprime la quantité de courant (I), transformée en chaleur ou en mouvement. Sur un barrage, elle correspond à l’énergie produite par la turbine.

La puissance P est le produit de la tension V en volts et de l'intensité I en ampères.

$$P = V \cdot I$$

Pour que ces électrons se déplacent tous dans un même sens, il faut qu'il y ait une différence du nombre d'électrons entre les deux extrémités du circuit électrique.
Pour maintenir cette différence du nombre d'électrons, on utilise un générateur (pile, accumulateur, alternateur...)
La différence de quantité d’électrons entre deux parties d’un circuit s’appelle la **différence de potentiel** et elle se mesure en **Volts (V)**.

Quelques ressources pour comprendre l’électricité:

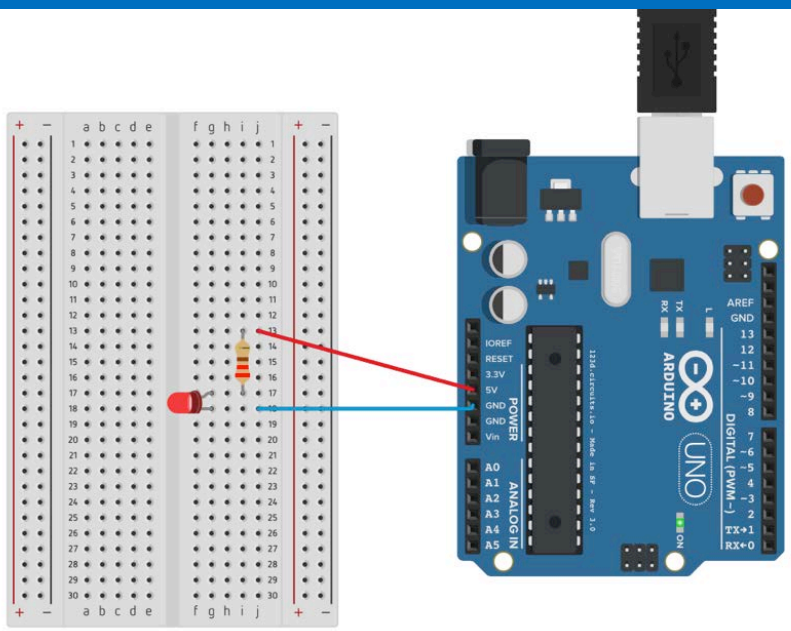
- *C’est pas sorcier* sur l’électricité: https://www.youtube.com/watch?v=efQW-ZmpyZs
- C'est pas sorcier sur le transport de l'électricité: https://www.youtube.com/watch?v=rMwuReV9DXk
- *C’est pas sorcier* sur les batteries et les piles: https://www.youtube.com/watch?v=mItO3l82Ic0
- Site pédagogique sur l'électricité Hydro-Québec: http://www.hydroquebec.com/comprendre/
- La bataille de l'électricité: https://www.youtube.com/watch?v=rbVqoJu6bQ8

Projet 1: le circuit électrique

Réalise le circuit suivant:

Note: la couleur des fils électriques importe peu, de même que leur position sur la platine d'expérimentation.

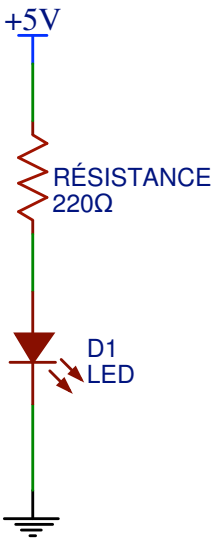
Circuit 1



Liste des composants

- 1 Led
- 1 résistance de 220 à 470Ω
- 2 câbles

Observations



Voici ce que cela donne au niveau du schéma électronique.

Lorsqu'on branche l'Arduino à l'ordinateur, via le câble USB, il y a 50% de chances que la LED s'allume. En effet, si elle ne s'allume pas, il faut tourner la LED dans l'autre sens. Sa particularité est que l'électricité ne peut la traverser que dans un sens.

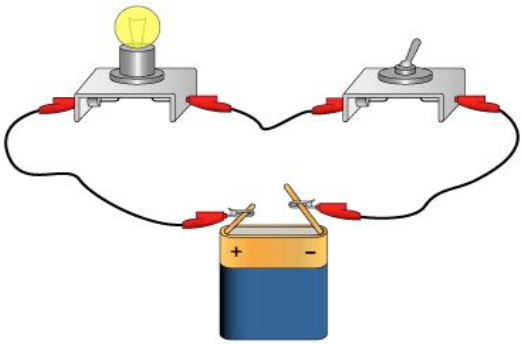
Le circuit électrique

Pour avoir un courant électrique, on a besoin d'une **source**, d'un **récepteur** et d'un **chemin**.

- La **source** fournit la force qui déplace les électrons: une pile, un accumulateur, un générateur…
- Le **récepteur** va utiliser la force de déplacement des électrons: une ampoule, un moteur…
- Le **chemin** conducteur permet aux électrons de se déplacer de la source au récepteur: souvent il s'agit de fils de cuivre.

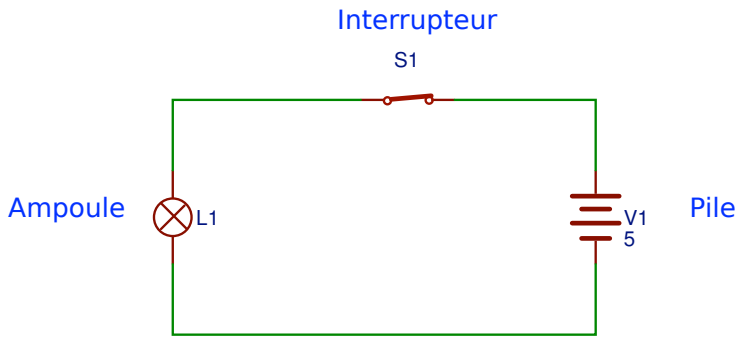
Ensemble, ils forment un circuit électrique. C'est ce qui a été réalisé lors de l'exercice: tu as relié une source d'électrons à la terre. Cela a crée une tension et donc un courant électrique: les électrons se sont déplacés. Leur déplacement a ainsi permis à la LED de s'allumer. L'Arduino ne sert qu'à l'alimentation électrique, comme une pile.

Une pile est constituée d'un milieu contenant de nombreux électrons en trop, et d'un second milieu en manque d'électrons. Quand on relie les deux pôles de la pile (le + et le -) avec un fil électrique (le conducteur), les électrons vont alors se déplacer du milieu riche en électrons vers le milieu pauvre. Si on place une lampe électrique entre les deux, le passage des électrons va générer de la lumière.



Circuit électrique

Voici le schéma électrique du circuit ci-dessus:

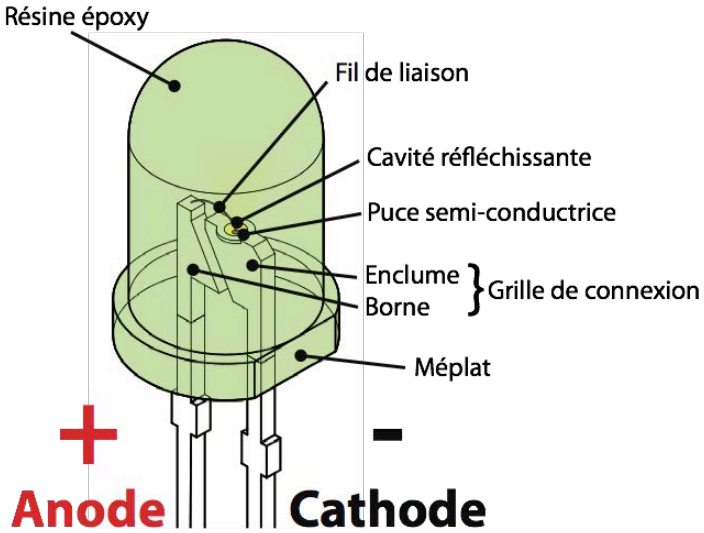


Lorsque l'interrupteur est enclenché, on dit que le circuit est **fermé**. Les électrons vont alors se déplacer et l'énergie de ce déplacement pourra être exploitée pour allumer une lampe ou faire fonctionner un moteur, par exemple.

Lorsque l'interrupteur est déclenché, on dit que le circuit est **ouvert**. Le pôle positif n'étant alors plus relié au pôle négatif, les électrons ne peuvent plus se déplacer.

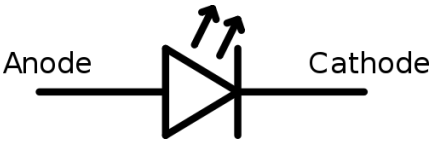
Les diodes

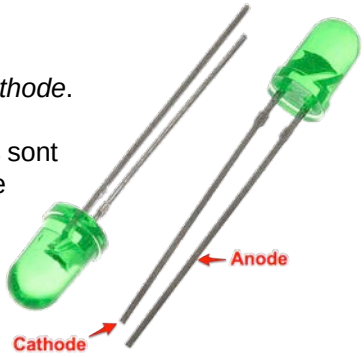
On l'a vu: le courant électrique se déplace dans un sens; *il a un sens de déplacement*. Il est possible de remplacer l'ampoule par une diode électroluminescente, aussi appelée LED[8]. **Elle a la particularité de ne laisser passer le courant électrique que dans un sens**.



Le courant électrique ne peut traverser la diode que dans le sens de *l'anode* vers la *cathode*.

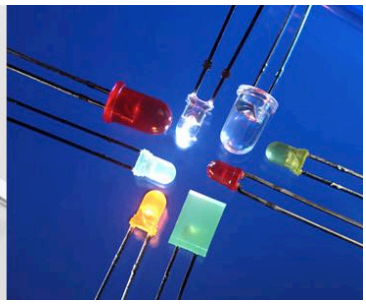
On reconnaît l'anode, car il s'agit de la broche la plus longue. Lorsque les deux broches sont de même longueur, on peut distinguer l'anode de la cathode, par un méplat du côté de cette dernière. Le symbole de la LED est le suivant:





Attention: le courant produit par l'Arduino est trop important pour y brancher directement une LED dessus. **L'utilisation d'une résistance est obligatoire, pour ne pas griller la LED**.

En utilisant divers matériaux semi-conducteurs, on fait varier la couleur de la lumière émise par la LED. Il existe enfin une grande variété de formes de LEDs.

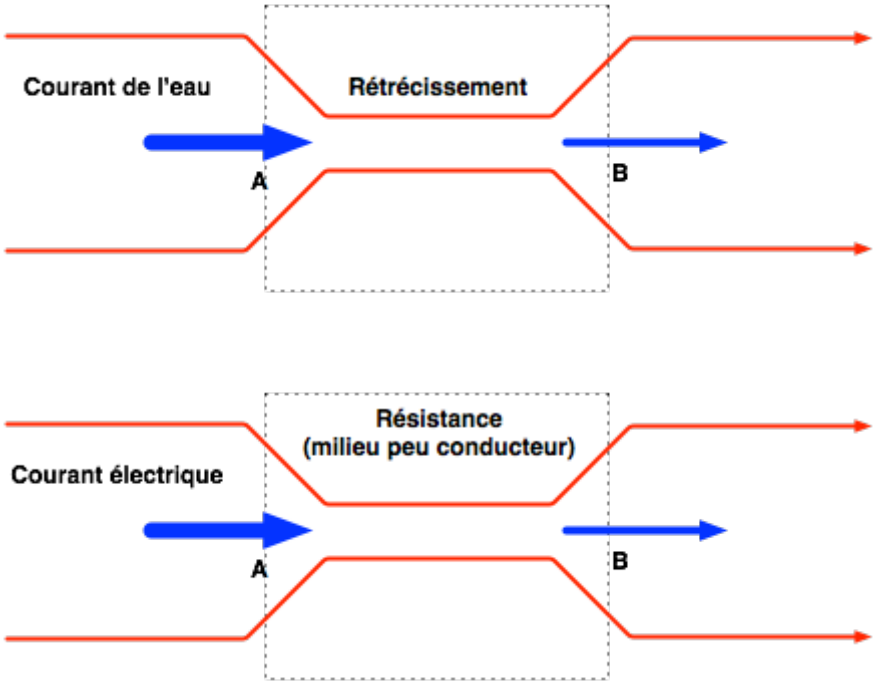


[8] http://fr.wikipedia.org/wiki/Diode_électroluminescente

Les résistances

Une **résistance** est un composant électronique ou électrique dont la principale caractéristique est d'opposer une plus ou moins grande résistance (mesurée en ohms: Ω) à la circulation du courant électrique.

On peut alors comparer, le débit d'eau au courant électrique **I** (qui est d'ailleurs le débit d'électrons), la différence de pression à la différence de potentiel électrique (qui est la tension **U**) et, enfin, le rétrécissement à la résistance **R**.



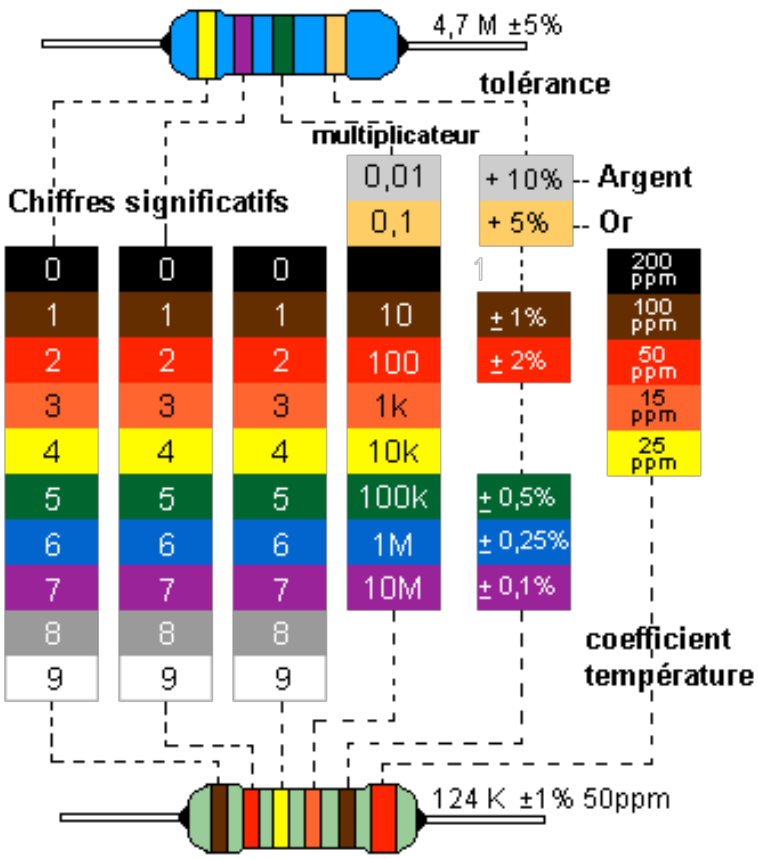
Ainsi, pour une tension fixe, plus la résistance est faible, plus le courant la traversant est fort. Cette proportion est vérifiée par la loi d'Ohm:

$$U = R \cdot I \; et \; donc \; R = \frac{U}{I} \; et \; I = \frac{U}{R}$$

Une résistance est un milieu peu conducteur; les électrons peinent à s'y déplacer. Leur énergie se dissipe alors en général sous forme de chaleur. C'est ce principe utilisé pour les bouilloires électriques ou les ampoules à filament.

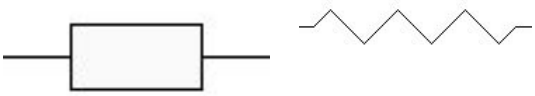
La valeur de la résistance se mesure en Ohms (Ω). La valeur d'une résistance est déterminée par ses bandes de couleurs.

Il est aussi possible d'utiliser une résistance comme réducteur de tension. Il suffit en effet de récupérer une source de courant électrique avant et après la résistance.



Outre le tableau ci-dessus, on peut s'aider du petit mode d'emploi qui se trouve ici:
http://www.apprendre-en-ligne.net/crypto/passecret/resistances.pdf

La résistance est schématisée de ces deux manières (européenne à gauche et américaine à droite):

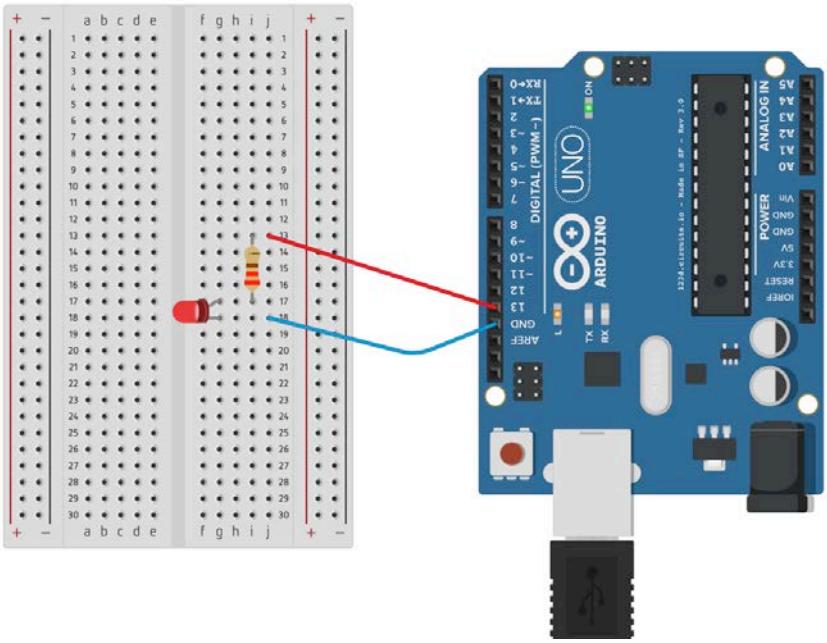


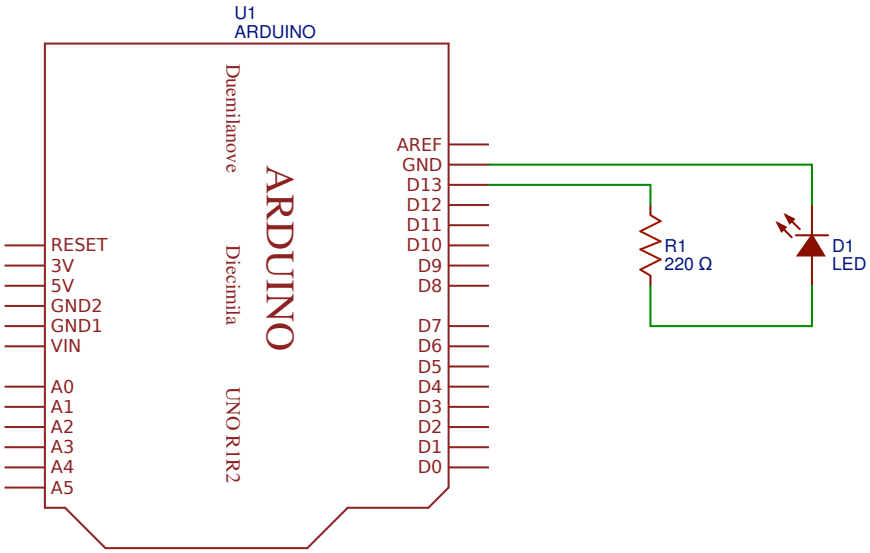
Un calculateur de valeurs de résistances est disponible à cette adresse: http://edurobot.ch/resistance/

Projet 2: faire clignoter une LED

Comme premier programme, nous allons faire clignoter une LED D1, connectée sur la broche 13. Voici le schéma de câblage:

Circuit 2





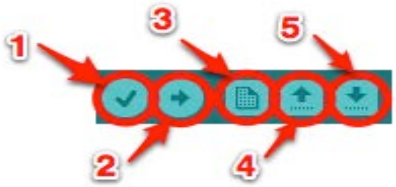
Liste des composants

- 1 Led
- 1 résistance de 220 à 470 Ω
- 2 câbles

Le logiciel Arduino IDE

La programmation se fait dans le logiciel Arduino IDE:

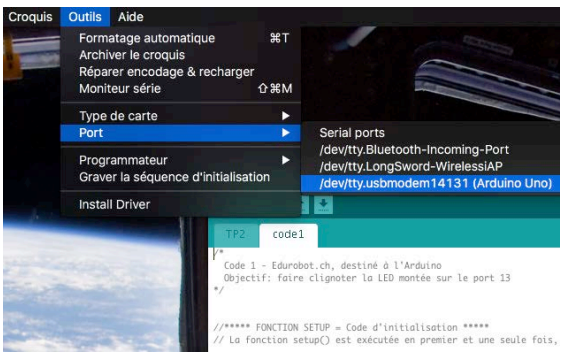
Le menu

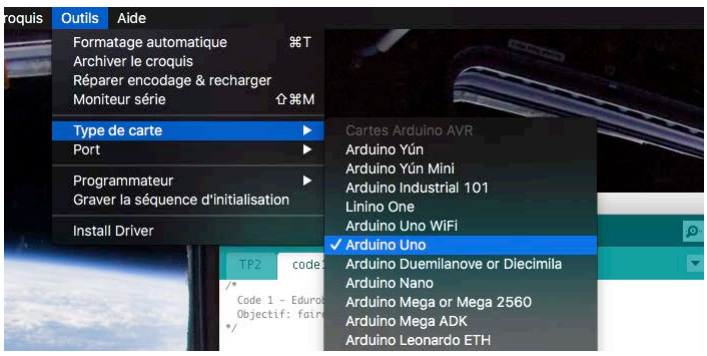


- Bouton 1 : Ce bouton permet de vérifier le programme, il actionne un module qui cherche les erreurs dans le programme
- Bouton 2 : Envoi du programme sur l'Arduino
- Bouton 3 : Créer un nouveau fichier
- Bouton 4 : Ouvrir un fichier existant
- Bouton 5 : Enregistrer un fichier

Commençons tout de suite par un petit code. Nous l'analyserons ensuite.

Une fois le code écrit (ou collé) dans la fenêtre de programmation, il faut l'envoyer sur l'Arduino. Pour cela, après avoir connecté l'Arduino à l'ordinateur, il faut sélectionner le port (*tty_usbmodemXXXXX*) et le type de carte (*Arduino Uno*, dans notre cas). Ces deux réglages sont dans le menu *Outils*.





Cliquer enfin sur le bouton *téléverser (upload)*.

Code 1: faire clignoter une LED sur la broche 13

```
/*
  Code 1 - Edurobot.ch, destiné à l'Arduino
  Objectif: faire clignoter la LED montée sur la broche 13
*/


//***** FONCTION SETUP = Code d'initialisation *****
// La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme


void setup()       // début de la fonction setup()

{
  pinMode(13, OUTPUT);    // Initialise la broche 13 comme sortie
  Serial.begin(9600);     // Ouvre le port série à 9600 bauds
}                    // fin de la fonction setup()


//***** FONCTION LOOP = Boucle sans fin = coeur du programme *****
// la fonction loop() s'exécute sans fin en boucle aussi longtemps que l'Arduino est sous
tension

void loop()       // début de la fonction loop()

{
  digitalWrite(13, HIGH);     // Met la broche 13 au niveau haut = allume la LED
  delay(500);                 // Pause de 500ms
  digitalWrite(13, LOW);      // Met la broche 13 au niveau bas = éteint la LED
  delay(500);                 // Pause 500ms
}               // fin de la fonction loop()
```

Observations

Ce code permet de faire clignoter la LED D1 située sur la broche 13. Une LED, soudée sur l'Arduino et reliée à la broche 13 clignote elle aussi. La résistante R1 de 220Ω sert à abaisser la tension et éviter que la LED ne grille.

Débugger

Il y a de nombreuses erreurs possibles dans un programme Arduino. La lecture et la compréhension de ces erreurs dans Arduino IDE ne sont pas évidentes. Vous trouverez de précieux conseils ici:

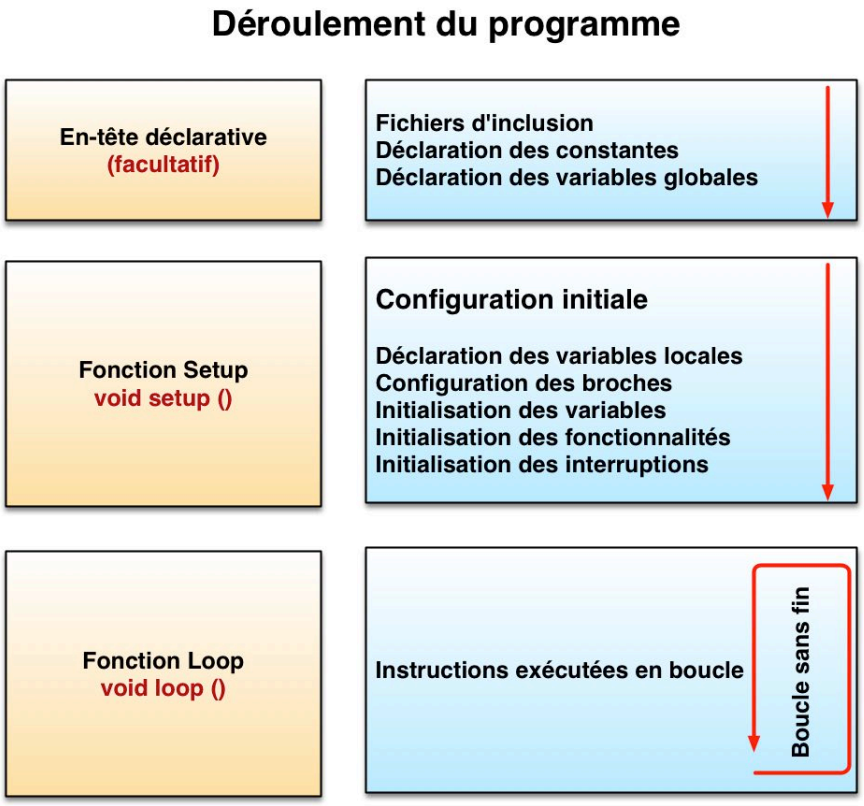
- https://www.didel.com/C/MiseAuPoint.pdf

Introduction au code

Le déroulement du programme

Le programme se déroule de la façon suivante :

1. Prise en compte des instructions de la partie déclarative
2. Exécution de la partie configuration (*fonction setup( )*),
3. Exécution de la boucle sans fin (*fonction loop ( )*): le code compris dans la boucle sans fin est exécuté indéfiniment.



Nous verrons petit à petit les divers éléments présents dans le schéma. L'important pour le moment est de savoir qu'un programme est divisé en trois parties: *en-tête déclarative*, *fonction setup* et *fonction loop*.

La suite va nous permettre d'entrer dans le code minimal: les fonctions setup et loop.

Le code minimal

Avec Arduino, nous devons utiliser un code minimal lorsqu'on crée un programme. Ce code permet de diviser le programme en deux parties.

```
void setup()
{
}

void loop()
{
}
```

Nous avons donc devant nous le code minimal qu'il faut insérer dans notre programme. Mais que peut-il bien signifier pour quelqu'un qui n'a jamais programmé ?

La fonction

Dans le code 1, se trouvent deux fonctions. Les fonctions sont en fait des *portions de code*.

Première fonction:

```
void setup()
{
}
```

Cette fonction setup() est appelée **une seule fois** lorsque le programme commence. C'est pourquoi c'est dans cette fonction que l'on va écrire le code qui n'a besoin d'être exécuté qu'une seule fois. On appelle cette fonction : "*fonction d'initialisation*". On y retrouvera la mise en place des différentes sorties et quelques autres réglages.

Une fois que l'on a initialisé le programme, il faut ensuite créer son "cœur", autrement dit le programme en lui-même.

Deuxième fonction:

```
void loop()
{
}
```

C'est donc dans cette fonction loop() que l'on va écrire le contenu du programme. Il faut savoir que cette fonction est appelée en permanence, c'est-à-dire qu'elle est exécutée une fois, puis lorsque son exécution est terminée, on la réexécute, encore et encore. On parle de *boucle infinie*.

Les instructions

Maintenant que nous avons vu la structure des fonctions, regardons ce qu'elles peuvent contenir.

Les instructions sont des lignes de code qui disent au programme : "fais ceci, fais cela..." Ce sont donc les ordres qui seront exécutés par l'Arduino. Il est très important de respecter exactement la syntaxe; faute de quoi, le code ne pourra pas être exécuté.

Les points virgules ;

Les points virgules terminent les instructions. Si par exemple on dit dans notre programme : "*appelle la fonction mangerLeChat*", on doit mettre un point virgule après l'appel de cette fonction.

Lorsque le code ne fonctionne pas, c'est souvent parce qu'il manque un point-virgule. Il faut donc être très attentif à ne pas les oublier!

Les accolades { }

Les accolades sont les "*conteneurs*" du code du programme. Elles sont propres aux fonctions, aux conditions et aux boucles. Les instructions du programme sont écrites à l'intérieur de ces accolades.
Pour ouvrir une accolade sur Mac, taper *alt-8* et *alt-9* pour la fermer.

Les commentaires

Les commentaires sont des lignes de codes qui seront ignorées par le programme. Elles ne servent en rien lors de l'exécution du programme. Ils permettent d'annoter et de commenter le programme.

Ligne unique de commentaire :

```
//cette ligne est un commentaire sur UNE SEULE ligne
```

Ligne ou paragraphe sur plusieurs lignes :

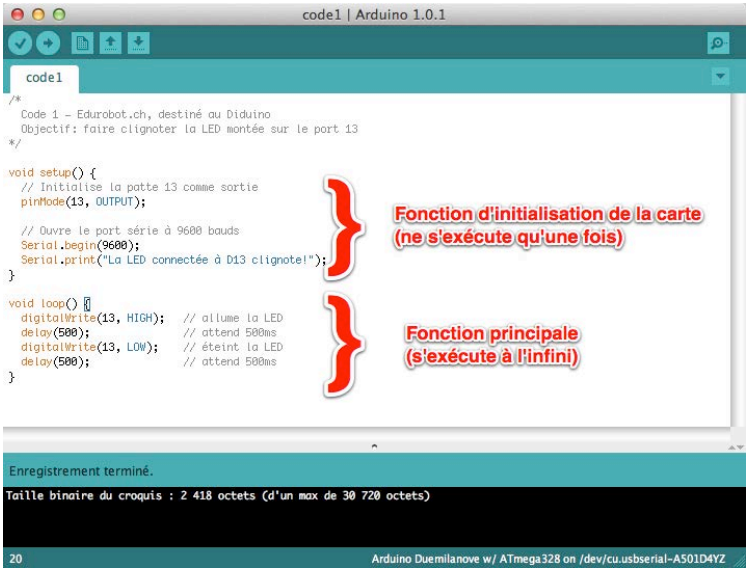
```
/*cette ligne est un commentaire, sur PLUSIEURS lignes
 qui sera ignoré par le programme, mais pas par celui qui lit le code ;) */
```

Les accents

Il est formellement interdit de mettre des accents en programmation! Sauf dans les commentaires...

Analyse du code 1

Revenons maintenant à notre code.



- La ligne *pinMode(13, OUTPUT);* initialise la broche 13 du microcontrôleur comme sortie, c'est-à-dire que des données seront envoyées depuis le microcontrôleur vers cette broche (on va envoyer de l'électricité).
- La ligne *Serial.begin(9600);* initialise le port série qui permet à l'Arduino d'envoyer et de recevoir des informations à l'ordinateur. C'est recommandé, mais cela fonctionne aussi sans.
- Avec l'instruction *digitalWrite(13, HIGH);*, le microcontrôleur connecte la broche D13 au +5V ce qui a pour effet d'allumer la LED (de l'électricité sort de la broche D13).
- L'instruction *delay(500);* indique au microcontrôleur de ne rien faire pendant 500 millisecondes, soit ½ seconde.
- Avec l'instruction *digitalWrite(13, LOW);*, le microcontrôleur connecte la broche D13 à la masse (Gnd) ce qui a pour effet d'éteindre la LED (on coupe l'alimentation en électricité).
- L'instruction *delay(500);* indique au microcontrôleur à nouveau de ne rien faire pendant 500ms soit ½ seconde.
- Le résultat est donc que la LED s'allume pendant ½ seconde, puis s'éteint pendant une ½ seconde puis s'allume pendant ½ seconde... elle clignote donc.

Profitons maintenant pour voir ce que signifie le terme *Output*. Il s'agit de préciser si la broche est une entrée ou une sortie. En effet, le microcontrôleur a la capacité d'utiliser certaines de ses broches en entrée (INPUT) ou en sortie (*OUTPUT*). Il suffit simplement d'interchanger une ligne de code pour dire qu'il faut utiliser une broche en entrée (récupération de données) ou en sortie (envoi de données).

Cette ligne de code doit se trouver dans la fonction *setup()*. La fonction est *pinMode()*, comme dans l'exemple ci-dessous:

```
void setup()
{
  pinMode(13, OUTPUT);
  Serial.begin(9600);
}
```

Modifions le code

Faisons varier les valeurs de l'instruction *delay* et modifions le code selon les exemples ci-dessous.

Essai 1:

```
digitalWrite(13, HIGH);
delay(50);
digitalWrite(13, LOW);
delay(50);
```

Essai 2:

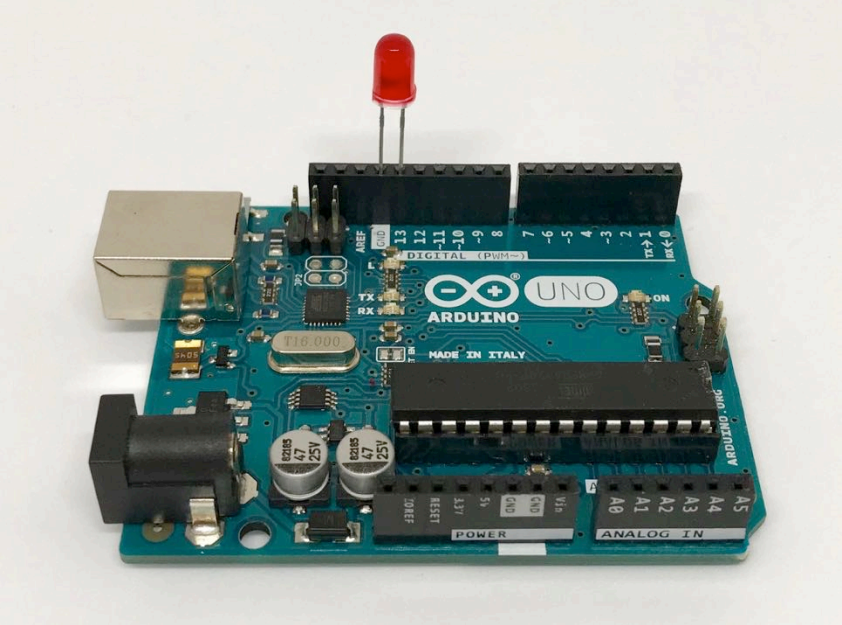
```
digitalWrite(13, HIGH);
delay(500);
digitalWrite(13, LOW);
delay(2000);
```

Essai 3:

```
digitalWrite(13, HIGH);
delay(50);
digitalWrite(13, LOW);
delay(50);
digitalWrite(13, HIGH);
delay(1000);
digitalWrite(13, LOW);
delay(1000);
```

Astuce:

Comme il y a déjà une résistance et une LED soudée à la l'Arduino et branchées sur la broche 13, il est possible d'installer directement une LED entre la broche et la terre (GND).



Exercice: lancer un SOS

En code morse, les lettres sont codées à l'aide d'une succession d'impulsions courtes et longues. On utilise le code morse pour transmettre un SOS, par exemple sous forme de signaux lumineux avec une lampe de poche.

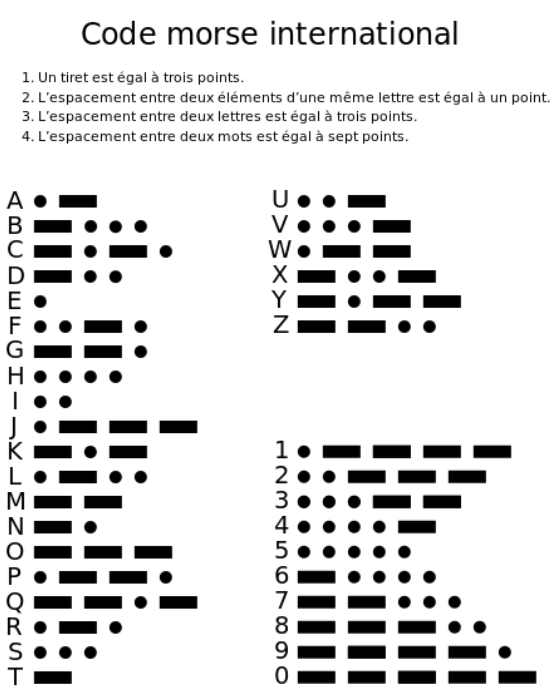
L'exercice consiste à coder l'Arduino et sa LED pour qu'il transmette un signal de SOS lumineux. Voici à quoi ressemble un SOS en morse:

■ ■ ■ ■■■ ■■■ ■■■ ■ ■ ■

Attention: il n'y a pas d'espace entre les lettres d'un SOS.

Une vidéo avec le résultat est visible ici:

- https://www.scolcast.ch/episode/arduino-sos



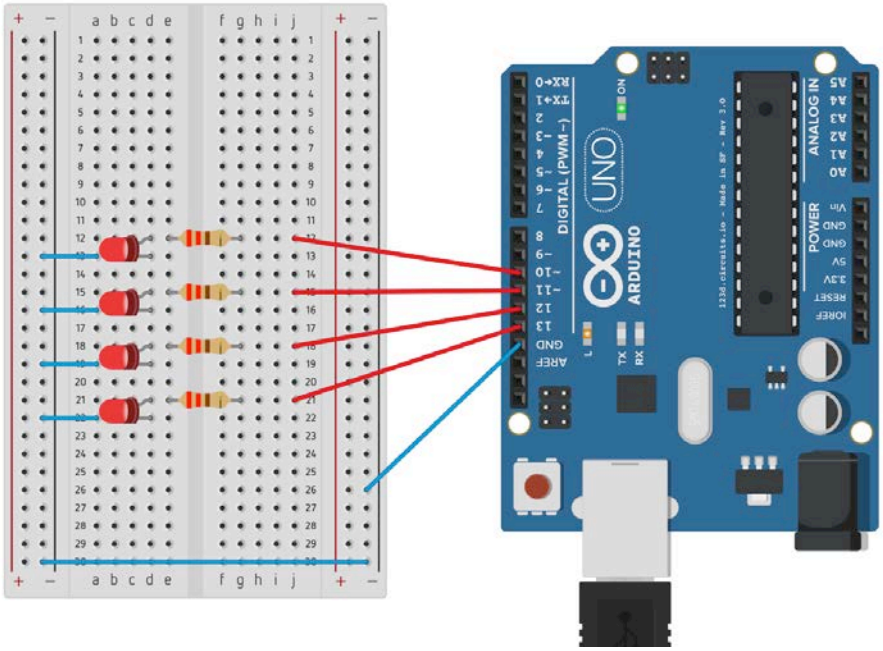
Code de l'exercice SOS

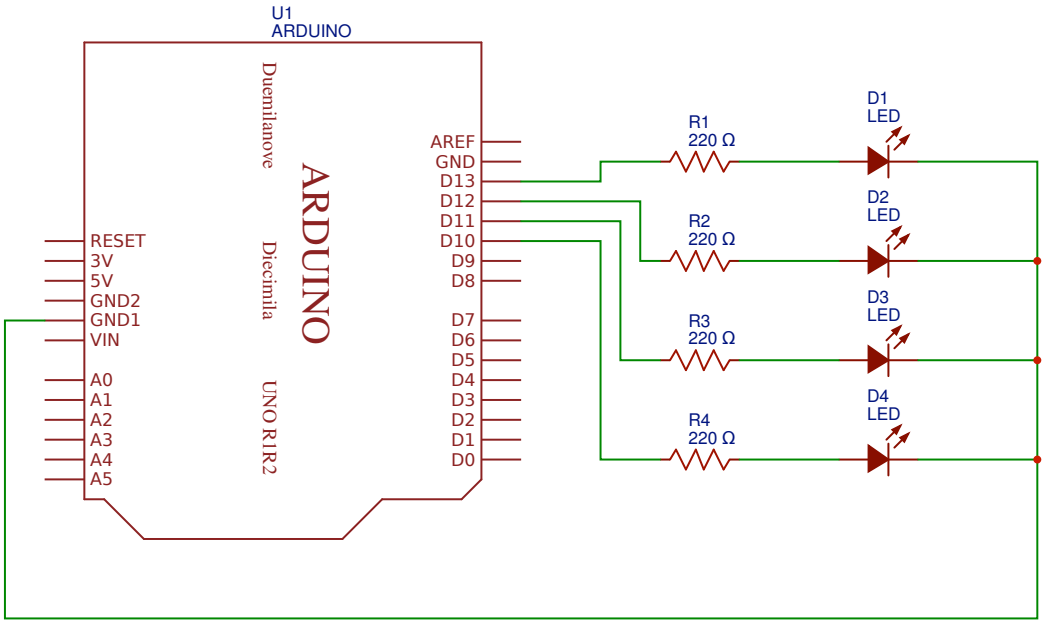
```
/*
  Exercice Code SOS - Edurobot.ch, destiné à l'Arduino
  Objectif: faire clignoter la LED montée sur la broche 13 pour lancer un SOS
*/

//***** FONCTION SETUP = Code d'initialisation *****
// La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme


void setup()       // début de la fonction setup()
{
  pinMode(13, OUTPUT);    // Initialise la broche 13 comme sortie
  Serial.begin(9600);     // Ouvre le port série à 9600 bauds
}                // fin de la fonction setup()


//***** FONCTION LOOP = Boucle sans fin = coeur du programme *****
// la fonction loop() s'exécute sans fin en boucle aussi longtemps que l'Arduino est sous
tension

void loop()       // début de la fonction loop()

{
  digitalWrite(13, HIGH);     // Met la broche 13 au niveau haut = allume la LED
  delay(200);                 // Pause de 200ms
  digitalWrite(13, LOW);      // Met la broche 13 au niveau bas = éteint la LED
  delay(200);                 // Pause 200ms

  digitalWrite(13, HIGH);     // Met la broche 13 au niveau haut = allume la LED
  delay(200);                 // Pause de 200ms
  digitalWrite(13, LOW);      // Met la broche 13 au niveau bas = éteint la LED
  delay(200);                 // Pause 200ms

  digitalWrite(13, HIGH);     // Met la broche 13 au niveau haut = allume la LED
  delay(200);                 // Pause de 200ms
  digitalWrite(13, LOW);      // Met la broche 13 au niveau bas = éteint la LED
  delay(200);                 // Pause 200ms

  digitalWrite(13, HIGH);     // Met la broche 13 au niveau haut = allume la LED
  delay(600);                 // Pause de 600ms
  digitalWrite(13, LOW);      // Met la broche 13 au niveau bas = éteint la LED
  delay(200);                 // Pause 200ms

  digitalWrite(13, HIGH);     // Met la broche 13 au niveau haut = allume la LED
  delay(600);                 // Pause de 600ms
  digitalWrite(13, LOW);      // Met la broche 13 au niveau bas = éteint la LED
  delay(200);                 // Pause 200ms

  digitalWrite(13, HIGH);     // Met la broche 13 au niveau haut = allume la LED
  delay(600);                 // Pause de 600ms
  digitalWrite(13, LOW);      // Met la broche 13 au niveau bas = éteint la LED
  delay(200);                 // Pause 200ms

  digitalWrite(13, HIGH);     // Met la broche 13 au niveau haut = allume la LED
  delay(200);                 // Pause de 200ms
  digitalWrite(13, LOW);      // Met la broche 13 au niveau bas = éteint la LED
  delay(200);                 // Pause 200ms

  digitalWrite(13, HIGH);     // Met la broche 13 au niveau haut = allume la LED
  delay(200);                 // Pause de 200ms
  digitalWrite(13, LOW);      // Met la broche 13 au niveau bas = éteint la LED
  delay(200);                 // Pause 200ms

  digitalWrite(13, HIGH);     // Met la broche 13 au niveau haut = allume la LED
  delay(200);                 // Pause de 200ms
  digitalWrite(13, LOW);      // Met la broche 13 au niveau bas = éteint la LED
  delay(1000);                 // Pause 1000ms
}            // fin de la fonction loop()
```

Projet 3: faire clignoter quatre LEDs

Voici le montage à réaliser. Les LEDs sont connectées aux broches 10 à 13.

Circuit 3





Liste des composants

- 4 LEDs
- 4 résistances de 220 à 470Ω
- 6 câbles

Code 2: faire clignoter 4 LEDs

Ce code fait clignoter les 4 LEDs en même temps.

```
/*
  Code 2 - Edurobot.ch, destiné à l'Arduino
  Objectif: faire clignoter les 4 LEDs montées sur les broches 10 à 13
*/


//***** FONCTION SETUP = Code d'initialisation *****
// La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme


void setup()       // début de la fonction setup()

{

  pinMode(10, OUTPUT);    // Initialise la broche 10 comme sortie
  pinMode(11, OUTPUT);    // Initialise la broche 11 comme sortie
  pinMode(12, OUTPUT);    // Initialise la broche 12 comme sortie
  pinMode(13, OUTPUT);    // Initialise la broche 13 comme sortie

  Serial.begin(9600);     // Ouvre le port série à 9600 bauds

}         // fin de la fonction setup()


//***** FONCTION LOOP = Boucle sans fin = coeur du programme *****
// la fonction loop() s'exécute sans fin en boucle aussi longtemps que l'Arduino est sous
//tension


void loop()       // début de la fonction loop()

{

  digitalWrite(10, HIGH);     // Met la broche 10 au niveau haut = allume la LED
  digitalWrite(11, HIGH);     // Met la broche 11 au niveau haut = allume la LED
  digitalWrite(12, HIGH);     // Met la broche 12 au niveau haut = allume la LED
  digitalWrite(13, HIGH);     // Met la broche 13 au niveau haut = allume la LED
  delay(500);  // Pause de 500ms
  digitalWrite(10, LOW);      // Met la broche 10 au niveau bas = éteint la LED
  digitalWrite(11, LOW);      // Met la broche 11 au niveau bas = éteint la LED
  digitalWrite(12, LOW);      // Met la broche 12 au niveau bas = éteint la LED
  digitalWrite(13, LOW);      // Met la broche 13 au niveau bas = éteint la LED
  delay(500);                 // Pause 500ms

}         // fin de la fonction loop()
```

Code 3: réaliser un chenillard à 4 LEDs.

L'objectif est d'allumer une LED après l'autre, selon le tableau suivant :

Séquence	LED 1	LED 2	LED 3	LED 4
1	x			
2		x		
3			x	
4				x

```
/*
  Code 3 - Edurobot.ch, destiné à l'Arduino
  Objectif: faire un chenillard à 4 LEDs montées sur les broches 10 à 13
*/

void setup()       // début de la fonction setup()

{
  pinMode(10, OUTPUT);    // Initialise la broche 10 comme sortie
  pinMode(11, OUTPUT);    // Initialise la broche 11 comme sortie
  pinMode(12, OUTPUT);    // Initialise la broche 12 comme sortie
  pinMode(13, OUTPUT);    // Initialise la broche 13 comme sortie

  Serial.begin(9600);     // Ouvre le port série à 9600 bauds
}         // fin de la fonction setup()


void loop()       // début de la fonction loop()

{
  digitalWrite(10, HIGH);     // Met la broche 10 au niveau haut = allume la LED
  digitalWrite(11, LOW);      // Met la broche 11 au niveau bas = éteint la LED
  digitalWrite(12, LOW);      // Met la broche 12 au niveau bas = éteint la LED
  digitalWrite(13, LOW);      // Met la broche 13 au niveau bas = éteint la LED

  delay(100);  // Pause de 100ms

  digitalWrite(10, LOW);      // Met la broche 10 au niveau bas = éteint la LED
  digitalWrite(11, HIGH);     // Met la broche 11 au niveau haut = allume la LED

  delay(100);  // Pause de 100ms

  digitalWrite(11, LOW);      // Met la broche 11 au niveau bas = éteint la LED
  digitalWrite(12, HIGH);     // Met la broche 12 au niveau haut = allume la LED

  delay(100);  // Pause de 100ms

  digitalWrite(12, LOW);      // Met la broche 12 au niveau bas = éteint la LED
  digitalWrite(13, HIGH);     // Met la broche 13 au niveau haut = allume la LED

  delay(100);  // Pause de 100ms

  digitalWrite(13, LOW);      // Met la broche 13 au niveau bas = éteint la LED
  digitalWrite(12, HIGH);     // Met la broche 12 au niveau haut = allume la LED

  delay(100);  // Pause de 100ms

  digitalWrite(12, LOW);      // Met la broche 12 au niveau bas = éteint la LED
  digitalWrite(11, HIGH);     // Met la broche 11 au niveau haut = allume la LED

  delay(100);  // Pause de 100ms
}         // fin de la fonction loop
```

Projet 4: introduction aux variables

Une variable, qu'est-ce que c'est ?

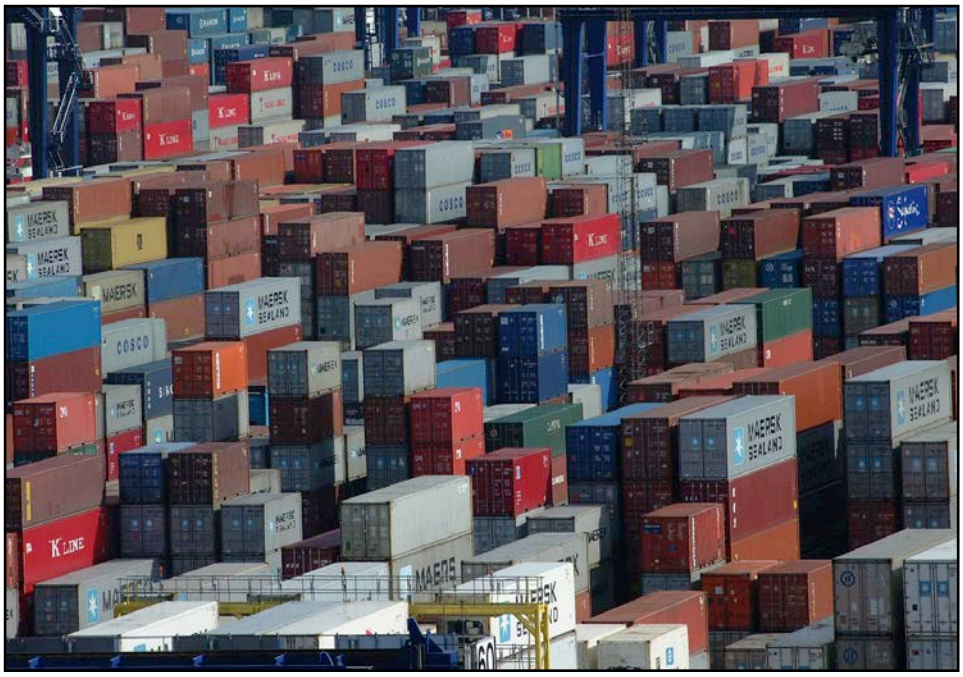
Imaginons un nombre dont nous devons nous souvenir. Ce nombre est stocké dans un espace de stockage de la mémoire vive (RAM) du microcontrôleur. Chaque espace de stockage est identifié de manière unique.

Le nombre stocké a la particularité de *changer de valeur*. En effet, la variable n'est que le conteneur. Son contenu va donc pouvoir être modifié. Et ce conteneur va être stocké dans une case de la mémoire. Si on matérialise cette explication par un schéma, cela donnerait :

nombre ➡ *variable* ➡ *mémoire*

le symbole "➡" signifiant : "est contenu dans..."

Imaginons que nous stockons le nombre dans un container (la variable). Chaque container est lui, déposé dans un espace bien précis, afin de le retrouver. Chaque container (variable) est aussi identifié par un nom unique.



Le nom d'une variable

Le nom de variable n'accepte que l'alphabet alphanumérique ([a-z], [A-Z], [0-9]) et _ (underscore). Il est unique; il ne peut donc pas y avoir deux variables portant le même nom.

Définir une variable

Imaginons que nous voulons stocker le nombre 4 dans une variable. Il tiendrait dans un petit carton. Mais on pourrait le stocker dans un grand container. Oui... mais non! Un microcontrôleur, ce n'est pas un ordinateur 3GHz multicore, 8Go de RAM ! Ici, il s'agit d'un système qui fonctionne avec un CPU à 16MHz (soit 0,016 GHz) et 2 Ko de SRAM pour la mémoire vive. Il y a donc au moins deux raisons pour choisir ses variables de manière judicieuse :

1. La RAM n'est pas extensible, quand il y en a plus, y en a plus! Dans un même volume, on peut stocker bien plus de petits cartons de que gros containers. Il faut donc optimiser la place.

2. Le processeur est de type 8 bits (sur un Arduino UNO), donc il est optimisé pour faire des traitements sur des variables de taille 8 bits, un traitement sur une variable 32 bits prendra donc (beaucoup) plus de temps. Si les variables de la taille d'un container sont sur 32 bits, autant prendre un carton qui n'occupe que 8 bits quand la variable tient dedans!

Type de variable	Type de nombre stocké	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
int	entier	-32'768 à +32'767	16 bits	2 octets
long	entier	-2'147'483'648 à +2'147'483'647	32 bits	4 octets
char	entier	-128 à +127	8 bits	1 octet
float	décimale	-3.4×10^{38} à $+3.4 \times 10^{38}$	32 bits	4 octets
double	décimale	-3.4×10^{38} à $+3.4 \times 10^{38}$	32 bits	4 octets

Prenons maintenant une variable que nous allons appeler «x». Par exemple, si notre variable "x" ne prend que des valeurs décimales, on utilisera les types *int*, *long*, ou *char*. Si maintenant la variable "x" ne dépasse pas la valeur 64 ou 87, alors on utilisera le type *char*.

```
char x = 0;
```

Si en revanche x = 260, alors on utilisera le type supérieur (qui accepte une plus grande quantité de nombre) à char, autrement dit *int* ou *long*.

```
int x = 0;
```

Si à présent notre variable "x" ne prend jamais une valeur négative (-20, -78, ...), alors on utilisera un type *non-signé*. C'est à dire, dans notre cas, un *char* dont la valeur n'est plus de -128 à +127, mais de 0 à 255.

Voici le tableau des types non signés, on repère ces types par le mot *unsigned* (de l'anglais : non-signé) qui les précède :

Type de variable	Type de nombre stocké	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
unsigned char	entier non négatif	0 à 255	8 bits	1 octet
unsigned float	entier non négatif	0 à 65'535	16 bits	2 octets
unsigned double	entier non négatif	0 à 4'294'967'295	32 bits	4 octets

Définir les broches du microcontrôleur

Jusqu'à maintenant, nous avons identifié les broches du microcontrôleur à l'aide de leurs numéros, comme dans l'exemple suivant: *pinMode(13, OUTPUT);*. Cela ne pose pas de problème quand on a une ou deux LEDs connectées. Mais dès qu'on a des montages plus compliqués, cela devient difficile de savoir qui fait quoi. Il est donc possible de renommer chaque broche du microcontrôleur.

Premièrement, définissons la broche utilisée du microcontrôleur en tant que variable.

```
const int led1 = 13;
```

Le terme *const* signifie que l'on définit la variable comme étant constante. Par conséquent, on change la nature de la variable qui devient alors constante.
Le terme *int* correspond à un type de variable. Dans une variable de ce type, on peut stocker un nombre allant de -2147483648 à +2147483647, ce qui sera suffisant! Ainsi, la broche *13* s'appellera *led1*.

Nous sommes donc en présence d'une variable, nommée *led1*, qui est en fait une constante, qui peut prendre une valeur allant de -2147483648 à +2147483647. Dans notre cas, cette constante est assignée au nombre 13.
Concrètement, qu'est-ce que cela signifie? Observons la différence entre les deux codes.

Broche non-définie

```
void setup() {
  pinMode(13, OUTPUT);
// Initialise la broche 13 comme sortie
  Serial.begin(9600);
}

void loop() {
  digitalWrite(13, HIGH);
delay(500);
  digitalWrite(13, LOW);
delay(500);
}
```

Broche définie

```
const int led1= 13;
//La broche 13 devient led1

void setup() {
  pinMode(led1, OUTPUT);
// Initialise led1 comme broche de sortie
  Serial.begin(9600);
}

void loop() {
  digitalWrite(led1, HIGH);
delay(500);
digitalWrite(led1, LOW);
delay(500);
}
```

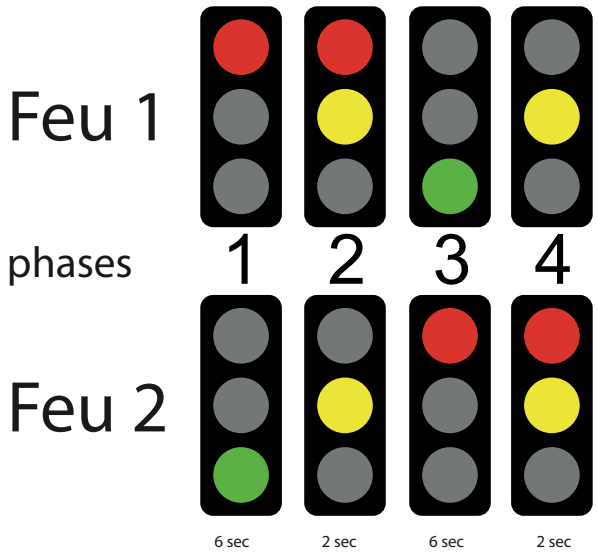
On peut trouver que de définir les broches allonge le code. Mais quand nous aurons de nombreuses broches en fonction, cela nous permettra de les identifier plus facilement. Ainsi, si nous avons plusieurs LED, nous pouvons les appeler *Led1*, *Led2*, *Led3*,... et si nous utilisons des LED de plusieurs couleurs, nous pourrons les appeler *rouge*, *vert*, *bleu*,...
Enfin (et surtout!), si on veut changer la broche utilisée, il suffit de corriger la variable au départ, sans devoir corriger tout le code.
Comme pour les variables, nous pouvons donner n'importe quel nom aux broches.

Projet 5: Les feux de circulation

Afin de mettre en pratique l'utilisation de variables, voici un petit exercice. On peut se baser sur le *code 3* pour débuter.

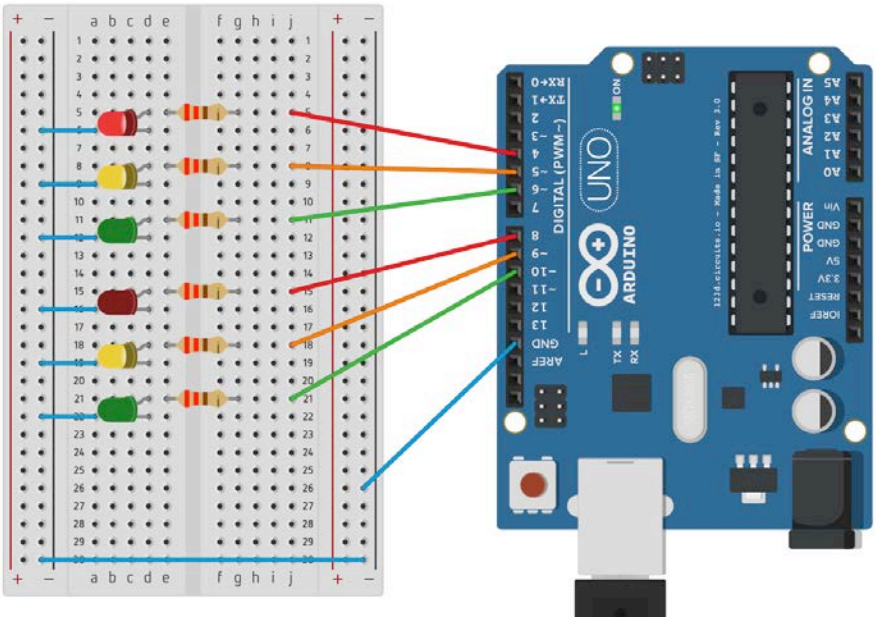
L'objectif de cet exercice est de créer deux feux de circulation et de les faire fonctionner de manière synchrone.

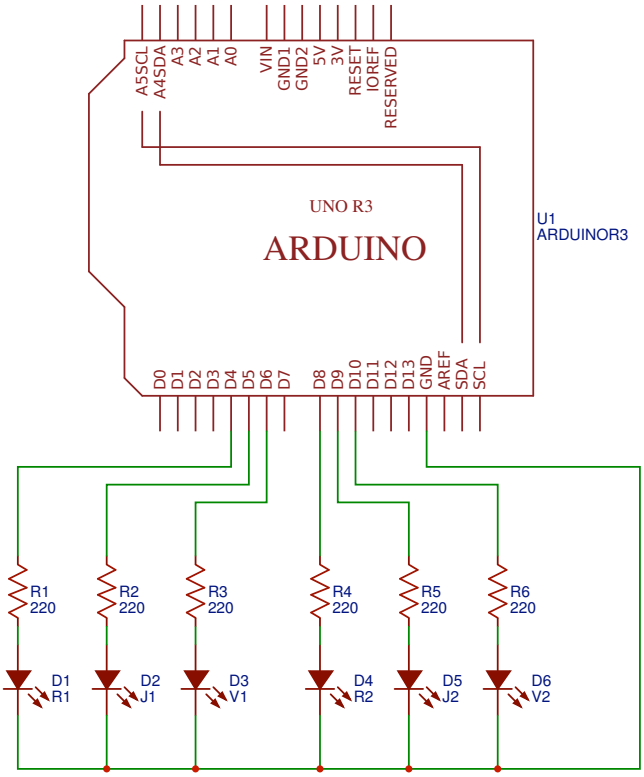
Voici les phases de deux feux de circulation que tu dois recréer:



Pour s'aider, vidéo de présentation est disponible ici: http://www.scolcast.ch/podcast/61/53-3067

Circuit 4





Liste des composants

- 2 LEDs rouges
- 2 LEDs jaunes ou oranges
- 2 LEDs vertes
- 6 résistances de 220 à 470Ω

Afin de faciliter l'identification de chaque LED, nous allons renommer les broches comme suit:

Feu 1:

- LED rouge connectée sur la broche 4 et renommée *R1*
- LED jaune connectée sur la broche 5 et renommée *J1*
- LED verte connectée sur la broche 6 et renommée *V1*

Feu 2:

- LED rouge connectée sur la broche 8 et renommée *R2*
- LED jaune connectée sur la broche 9 et renommée *J2*
- LED verte connectée sur la broche 10 et renommée *V2*

Enfin, nous utiliserons deux variables *timer1* et *timer2* pour définir les temps d'allumages. Avant de regarder le code à la page suivante, essaie de faire l'exercice seul.

Code 4: le feu de circulation

```
/*
  Code 4 - Edurobot.ch, destiné à l'Arduino
```

```
  Objectif: gérer des feux de circulation
*/

// On définit les variables pour chaque broche
//FEU 1
const int R1 = 4;      //La broche 4 devient le feu rouge 1
const int J1 = 5;      //La broche 3 devient le feu jaune 1
const int V1 = 6;      //La broche 2 devient le feu vert 1
//FEU2
const int R2 = 8;      //La broche 8 devient le feu rouge 2
const int J2 = 9;      //La broche 9 devient le feu jaune 2
const int V2 = 10;     //La broche 10 devient le feu vert 2
//TEMPS
int timer1 = 2000;     //Le temps est fixé à 2 secondes
int timer2 = 6000;     //Le temps est fixé à 6 secondes

void setup() {

//On initialise les sorties
pinMode(R1, OUTPUT);
pinMode(J1, OUTPUT);
pinMode(V1, OUTPUT);

pinMode(R2, OUTPUT);
pinMode(J2, OUTPUT);
pinMode(V2, OUTPUT);
}

void loop() {

//Phase 1: R1 et V2 sont allumés, R2, J1 et J2 sont éteints
digitalWrite(R2, LOW);     //R2 éteint
digitalWrite(J1, LOW);     //J1 éteint
digitalWrite(J2, LOW);     //J2 éteint
digitalWrite(R1, HIGH);    //R1 allumé
digitalWrite(V2, HIGH);    //V2 allumé
delay(timer2);              //durée: 6'000 millisecondes, soit 6 secondes

//Phase 2: R1, J1, J2 allumés et V2 éteint
digitalWrite(V2, LOW);     //V2 éteint
digitalWrite(J1, HIGH);    //J1 allumé
digitalWrite(J2, HIGH);    //J2 allumé
delay(timer1);               //durée: 2'000 millisecondes, soit 2 secondes

//Phase 3: R1, J1, J2 éteints et V1 et R2 allumés
digitalWrite(R1, LOW);     //R1 éteint
digitalWrite(J1, LOW);     //J1 éteint
digitalWrite(J2, LOW);     //J2 éteint
digitalWrite(V1, HIGH);    //V1 allumé
digitalWrite(R2, HIGH);    //R2 allumé
delay(timer2);

//Phase 4: V1 éteint et J1, J2 et R2 allumés
digitalWrite(V1, LOW);     //V1 éteint
digitalWrite(J1, HIGH);    //J1 allumé
digitalWrite(J2, HIGH);    //J2 allumé
digitalWrite(R2, HIGH);    //R2 allumé
delay(timer1);
}
```

Variantes

Variante 1

Il y a souvent un décalage entre le passage d'un feu au rouge et le passage au vert de l'autre feu. C'est en particulier le cas pour les feux de chantier. Cela permet aux voitures encore engagées dans la zone de chantier de la quitter, avant de laisser la place aux voitures venant en face.

Décrire les phases des feux et les programmer pour tenir compte du délai.

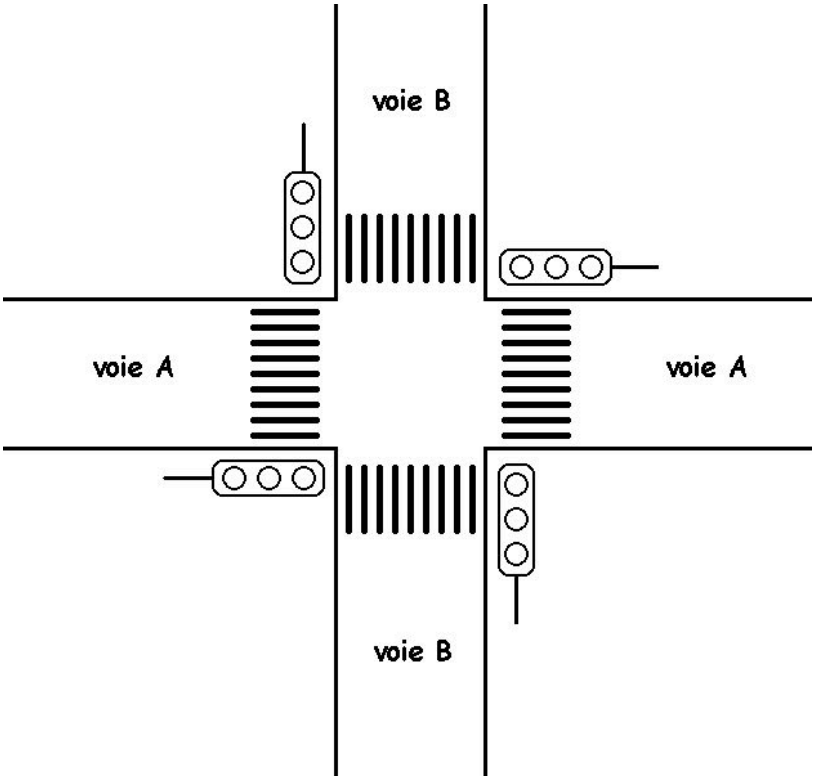
Variante 2

Intégrer un troisième feu, qui passe du rouge au vert en alternance avec les deux autres feux.

Réaliser le schéma électronique et programmer les feux.

Variante 3

Réaliser les feux pour un carrefour:



Projet 6: L’incrémentation

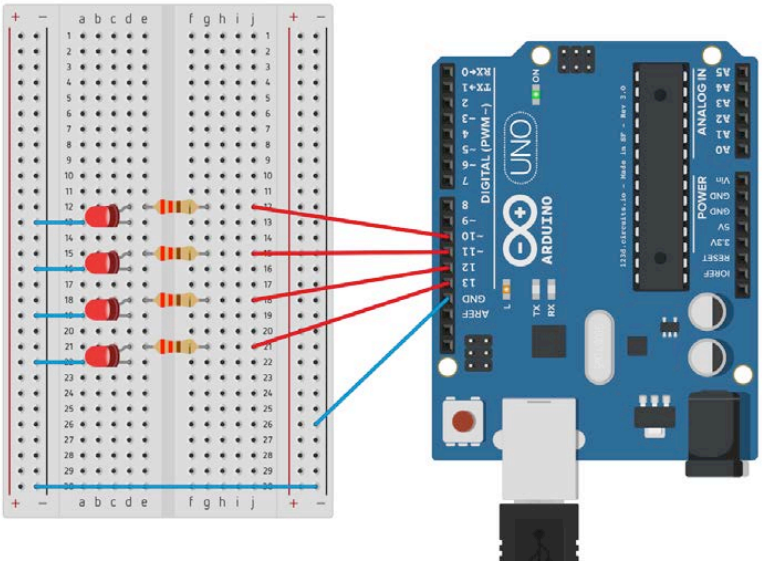
Il est possible d’appliquer aux valeurs des variables diverses opérations mathématiques. Commençons tout de suite par un petit exemple: *l’incrémentation*. Il s’agit simplement d’additionner 1 à la variable. A chaque fois qu’on répète le code, on prend la valeur de la variable et on y ajoute 1.

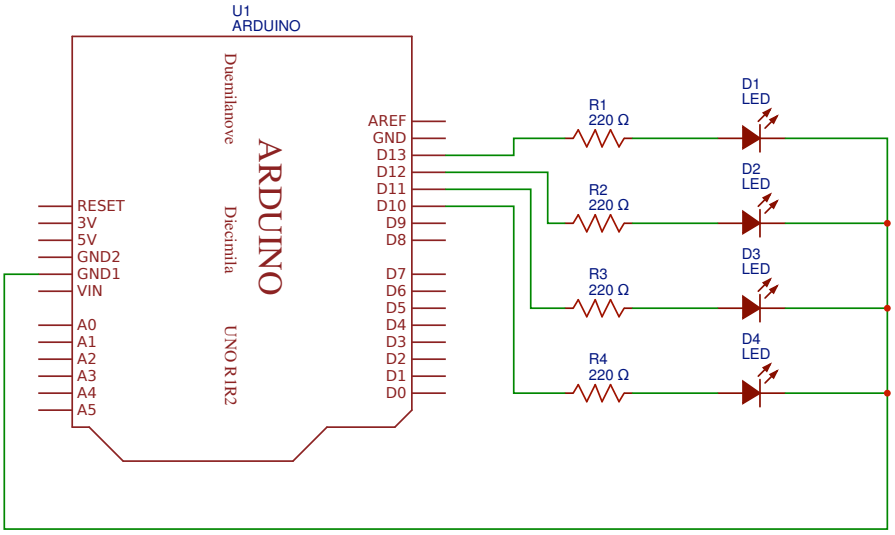
Cela se fait grâce à ce code (*var* étant une variable à choix): *var++;*

var++; revient à écrire : "var = var + 1;".

Le circuit sera des plus classiques: on va reprendre notre chenillard.

Circuit 5





Liste des composants:

- 4 LEDs rouges
- 4 résistances de 220 à 470Ω

Code 5: faire clignoter 10 fois la LED 13

```
/*
Code 5 - Edurobot.ch, destiné à l'Arduino
Objectif: faire clignoter 10 fois la LED montée sur le port 13
*/

//***** EN-TETE DECLARATIVE *****
// On déclare les variables, les constantes...

byte compteur;                 //On définit la variable "compteur"
const int led1= 13;     //On renomme la broche 10 en "led1"


//***** FONCTION SETUP = Code d'initialisation *****
// La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme

void setup()

{
  pinMode(led1, OUTPUT);       // Initialise la broche 13 comme sortie

  Serial.begin(9600);          // Ouvre le port série à 9600 bauds

// Exécute le programme entre accolades en partant de zéro et en incrémentant à chaque fois la
//valeur de +1: 0+1/2+1/3+1... jusqu'à ce que la variable "compteur" soit égale à 9 (plus //
// petit que 10).

for(compteur=0 ; compteur<10 ; compteur++)

  {                             // début du programme exécuté 10 fois
    digitalWrite(led1, HIGH); // allume la LED
    delay(500);               // attend 500ms
    digitalWrite(led1, LOW);  // éteint la LED
    delay(500);               // attend 500ms
  }
}                       // fin du programme exécuté 10 fois

void loop() {
}                       // vide, car programme déjà exécuté dans setup
```

Analyse du code[9]

Revenons à notre code et analysons-le.

La ligne *byte compteur;* permet de créer une variable appelée *compteur*. *Byte* indique le type de la variable, c'est-à-dire le type de données que l'on pourra stocker dans cette variable. Comme nous l'avons déjà vu, le type byte permet de stocker des valeurs comprises entre 0 et 255.

La ligne const int *led1= 13;* permet de créer une constante (une variable) nommée *led1* dans laquelle on stocke la valeur 13.

La ligne *for(compteur=0 ; compteur<10 ; compteur++)* sert à faire varier la variable compteur de 0 à 9 (en l'augmentant à chaque fois de 1: c'est ce que fait l'instruction *compteur++*)

Regardons cette ligne d'un peu plus près:

for(compteur=0 ; compteur<10 ; compteur++)

[9] Source: http://mediawiki.e-apprendre.net/index.php/Découverte_des_microcontrôleurs_avec_le_Diduino

La déclaration *for* est habituellement utilisée pour répéter un bloc d'instructions entourées de parenthèses. On l'utilise souvent avec un compteur incrémentiel, qui permet de terminer une boucle après un certain nombre de fois.

La suite, à savoir *compteur=0 ; compteur<10 ; compteur++* doit être compris comme suit: «la valeur de la variable *compteur* est comprise entre 0 et 9 (<10 signifie "plus petit que 10") et ajoute un à la valeur de *compteur* (c'est le *compteur++)».*

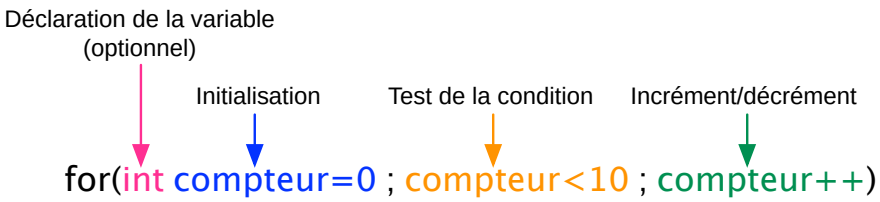
En conclusion, cette ligne signifie:

«Au commencement, la variable compteur est égale à zéro. On va exécuter le code en boucle. A chaque fois, on ajoute +1 à ta variable compteur, jusqu'à ce qu'on arrive à 9. A ce moment, on s'arrête.»

Le code qui est exécuté à chaque fois est celui qui est compris jusqu'à l'accolade *}*. Dans notre cas, c'est e code suivant qui est exécuté 10 fois:

```
    digitalWrite(Led1, HIGH);
    delay(500);
    digitalWrite(Led1, LOW);
    delay(500);
```

Voici les parties d'une déclaration *for:*



L'initialisation est effectuée en premier et une seule fois. A chaque passage de la boucle, la condition est testée. Si elle est "vrai", le contenu de la boucle for est exécuté (par exemple faire clignoter une LED), et l'incrément de la variable est réalisé. Ensuite, la condition est testée à nouveau. Dès que le résultat du test de la condition est "faux", la boucle s'arrête

Code 6: Réaliser un chenillard sur les broches 10 à 13 avec un *for*

Nous pouvons sans problème utiliser une incrémentation et un for pour réaliser le pinMode de Leds qui se suivent, par exemple pour réaliser un chenillard:

```
/*
  Code 6 - Edurobot.ch, destiné à l'Arduino
  Objectif: faire un chenillard à 4 LEDs montées sur les ports 10 à 13
*/

//***** EN-TETE DECLARATIVE *****

// Définition de la variable "temps"

int timer = 100;           // Durée, en millisecondes


//***** FONCTION SETUP = Code d'initialisation *****


void setup() {
```

```
// Déclaration des broches 10 à 13 à l'aide d'un for et d'un incrément.
  for (int thisPin = 10; thisPin < 14; thisPin++) {
    pinMode(thisPin, OUTPUT);
  }
}

void loop() {
  // boucle de la broche 10 à la broche 13:

for (int thisPin = 10; thisPin < 14; thisPin++) {   //Incrément faisant passer la variable
thisPin de 10 à 13
    digitalWrite(thisPin, HIGH);   //Allumer la LED
    delay(timer);                  //Durée
    digitalWrite(thisPin, LOW);    //Eteindre la LED
  }

  // boucle de la broche 13 à la broche 10:
  for (int thisPin = 13; thisPin >= 10; thisPin--) {  //Décrément faisant passer la variable
thisPin de 13 à 10
    digitalWrite(thisPin, HIGH);   //Allumer la LED
    delay(timer);                  //Durée
    digitalWrite(thisPin, LOW);    //Eteindre la LED;
  }
}
```

Analyse du code

Regardons maintenant un peu ce code, en particulier:

```
for (int thisPin = 10; thisPin < 14; thisPin++) {
    pinMode(thisPin, OUTPUT);
  }
```

Sa première particularité est la manière de définir les broches. Au lieu d'accumuler les *pinMode (xxx, OUTPUT);*, on créé une variable de type *int* qu'on appelle *thisPin*[10] et donc la valeur initiale est de *10*. Un *for* avec un incrément (++) permet de permet d'incrémenter la valeur de la variable de 10 à 14. Il suffit ensuite de remplacer dans le *pinMode* le numéro de la proche par la variable *thisPin*. Ainsi, les broches 10, 11, 12, 13 et 14 sont définies en output. Naturellement, cela ne fonctionne que si les broches utilisées se suivent (par exemple: 10, 11, 12, 13. Cela ne fonctionnera pas pour des broches 3, 5, 7, 8, 10).

Passons maintenant à la seconde partie:

```
for (int thisPin = 10; thisPin < 14; thisPin++) {
    digitalWrite(thisPin, HIGH);
    delay(timer);
    digitalWrite(thisPin, LOW);
  }
```

Comme pour la définition des broches en OUTPUT, on utilise la fonction *for* pour incrémenter la variable *thisPin* de 10 à 13. Ainsi, on allume les LEDs connectés sur les broches 10 à 13 l'une après l'autre.

Enfin dans la troisième partie du code, on va allumer les LEDs de la broche 13 à la broche 10 avec une fonction *for* et une décrémentation. Pour cela, on remplace le ++ par un --.

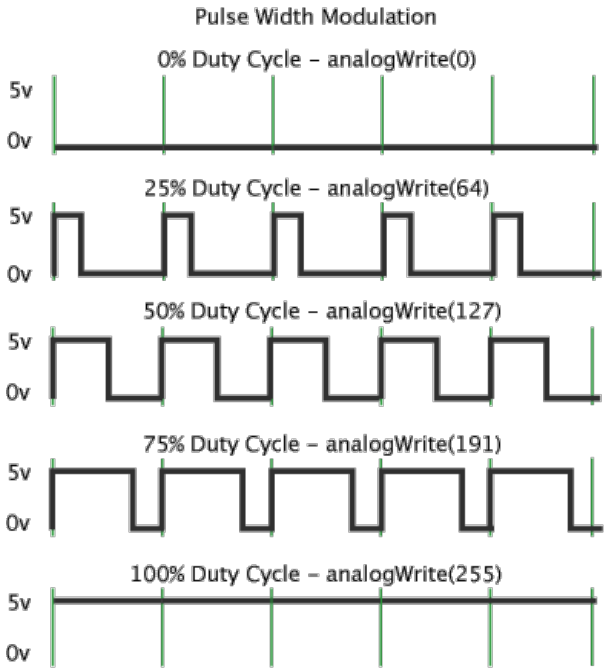
```
for (int thisPin = 13; thisPin >= 10; thisPin--) {
    digitalWrite(thisPin, HIGH);
    delay(timer);
    digitalWrite(thisPin, LOW);
```

[10] Mais on aurait aussi pu l'appeler *petite_fleur*...

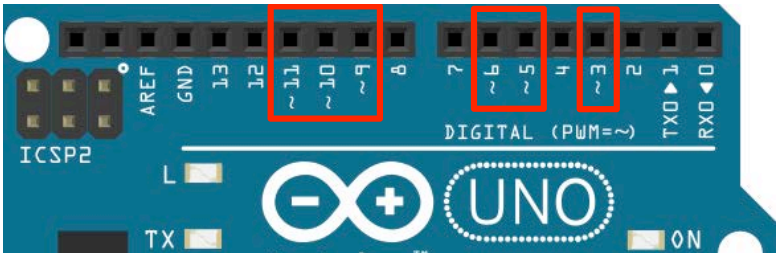
Projet 7: PWM, variation en douceur d'une LED

Le plus simple moyen de faire varier la luminosité d'une LED, c'est de faire varier le courant qui la traverse. Mais lorsqu'elle est branchée sur la broche d'un Arduino, ce n'est pas possible: les broches 1 à 13 sont en effet numériques. C'est-à-dire qu'elles n'ont que deux états: **0** ou **1**; allumé ou éteint. Alors pour faire varier la luminosité d'une LED, on va utiliser une fonction appelée PWM: *Pulse Width Modulation,* soit **modulation de largeur d'impulsions**. Il s'agit de faire varier les périodes hautes (allumé) et basses (éteint) des broches à grande fréquence. Ainsi, lors d'un cycle de 25% en position haute et 75% en position basse, la LED sera moins brillante que pour un cycle à 50%/50%.

En d'autre terme, cela signifie qu'on va faire clignoter très vite les LEDs; tellement vite que l'œil ne percevra qu'une lumière continue. Mais plus en augmentera la durée des périodes où la LED est éteinte, moins il y aura de lumière émise; et donc moins la LED semblera brillante.



Les broches capables de supporter du PWM sont identifiées par un "~". Il s'agit des broches 3, 5, 6, 9, 10,11.



Par distinction, au lieu d'utiliser l'instruction DigitalWrite, pour utiliser le PWM, on utilise AnalogWrite.

La valeur du PWM s'étend sur 256 paliers, de 0 (=0%) à 255 (=100%). On peut ainsi définir la valeur PWM souhaitée avec la formule suivante:

$$Valeur\ PWM = \frac{Pourcentage\ souhaité}{100} \cdot 255 \qquad Valeur\ en\ \% = \frac{Valeur\ PWM}{255} \cdot 100$$

Code 7: faire varier la luminosité d'une LED en modifiant la valeur PWM

```
/*
  Code 7 - Edurobot.ch, destiné à l'Arduino
  Objectif: faire varier la luminosité d'une LED sur la broche 10 en modifiant la valeur PWM
*/


//***** EN-TETE DECLARATIVE *****

int ledPin = 10;    //On renomme la broche 10 en "ledPin"
int timer = 100;   //On définit une durée de 0,1 seconde pour la variable timer

//***** SETUP  *****

void setup() {

pinMode(ledPin, OUTPUT);

}

//***** LOOP  *****

void loop() {

    // LED à 0%.
    analogWrite(ledPin, 0);
    delay(timer);

    // LED à 19.6%.
    analogWrite(ledPin, 50);
    delay(timer);

    // LED à 39.2%.
    analogWrite(ledPin, 100);
    delay(timer);

    // LED à 58.8%.
    analogWrite(ledPin, 150);
    delay(timer);

    // LED à 78.4%.
    analogWrite(ledPin, 200);
    delay(timer);

    // LED à 100%.
    analogWrite(ledPin, 255);
    delay(timer);
}
```

Analyse du code

Pas de surprise pour ce code, si ce n'est l'utilisation d'*analogWrite* en lieu et place de *digitalWrite*. La luminosité de la LED progresse par paliers de 50. Si on veut lisser la progression de la luminosité, il faut augmenter le nombre de paliers, ce qui va vite fortement alourdir le code. C'est la raison pour laquelle il est préférable d'utiliser une fonction for, comme nous le verrons dans le code suivant.

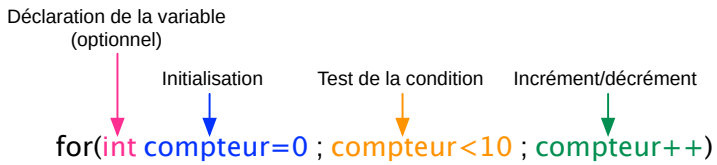
Code 8: faire varier la luminosité d'une LED en douceur

```
/*
  Code 8 - Edurobot.ch, destiné à l'Arduino
  Objectif: faire varier la luminosité d'une LED sur la broche 10
  Adapté de David A. Mellis et Tom Igoe
*/
```

```
//***** EN-TETE DECLARATIVE *****

int ledPin = 10;    //On renomme la broche 10 en "ledPin"

//***** SETUP *****

void setup() {

}

//***** LOOP *****

void loop() {

  // Variation du min au max par addition de 5 jusqu'à 256

  for (int fadeValue = 0 ; fadeValue <= 255; fadeValue += 5)
  {
    // Définition de la valeur de luminosité (de 0 à 255)
    analogWrite(ledPin, fadeValue);

    // Attente de 30 millisecondes entre chaque palier pour voir l'effet.
    delay(30);
  }

  // Variation du max au min par soustraction de 5 depuis 256

  for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -= 5)
  {
    // Définition de la valeur de luminosité (de 0 à 255)
    analogWrite(ledPin, fadeValue);

    // Attente de 30 millisecondes entre chaque palier pour voir l'effet.
    delay(30);
  }

}
```

Analyse du code

La fonction *for*, on commence à bien la connaître. Sinon, voici un petit rappel:



Dans notre code, nous commençons avec un for (int fadeValue = 0 ; fadeValue <= 255; fadeValue += 5).

On créé donc une variable *fadeValue* de type *int*, qui stocke le chiffre 0. Le test de la condition *plus petit ou égal à 255* (*fadeValue <= 255*) permet une sortie de la fonction dès qu'on atteint le nombre 256. Enfin; et c'est la nouveauté, on ne fait pas une incrémentation (++), mais on additionne 5 à *fadeValue* avec la formule *+=5*.

La commande *analogWrite(ledPin, fadeValue);* permet ainsi d'envoyer la valeur de *fadeValue* à la LED *ledPin*. Comme *fadeValue* augmente à chaque fois de +5, la luminosité de la LED augmente petit-à-petit. Enfin, le *delay* permet de configurer la vitesse d'augmentation de la luminosité. Chaque palier de 5 prend 30 millisecondes. Il faut 51 paliers de 5 pour passer de 0 à 255. Ainsi, il faudra $51 \cdot 30 = 1530$ millisecondes pour passer de la LED éteinte à la LED à sa luminosité maximum, soit environ une seconde et demie.

Ensuite, une fois la LED à sa luminosité maximum, il s'agit de baisser sa luminosité jusqu'à l'éteindre, grace au code for *(int fadeValue = 255 ; fadeValue >= 0; fadeValue -= 5)*. Le *+=5* est remplacé par un *-=5*, afin de soustraire à chaque fois 5 à la variable *fadeValue*.

Code 9: alternative pour faire varier la luminosité d'une LED

Avec le code 8, nous avons fait varier la luminosité d'une LED de 0 à 255 par paliers de 5. On peut tout aussi bien utiliser un incrément ++ pour arriver au même résultat. Voilà ce que cela donne:

```
/*
  Code 9 - Edurobot.ch, destiné à l'Arduino
  Objectif: faire varier la luminosité d'une LED sur la broche 10
*/


//***** EN-TETE DECLARATIVE *****

int ledPin = 10;    //On renomme la broche 10 en "ledPin"

//***** SETUP *****

void setup()
{
}

//***** LOOP *****

void loop()
{

  // Variation du min au max par incrément

   for (int fadeValue =0; fadeValue <= 255; fadeValue ++){
      analogWrite(ledPin, fadeValue);
      delay(10);
   }

  // Variation du max au min par décrément

   for (int fadeValue =255; fadeValue >= 0; fadeValue --){
      analogWrite(ledPin, fadeValue);
      delay(10);
   }
}
```

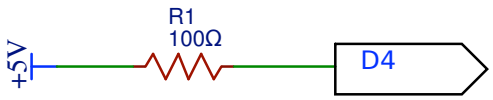
Projet 8: les inputs numériques

Jusqu'à maintenant, nous avons traité des outputs, c'est-à-dire qu'un signal sortait du microcontrôleur sous la forme d'un courant électrique *(HIGH)* ou de son absence *(LOW)* grâce à la commande DigitalWrite. On a utilisé jusqu'ici ce signal (un courant électrique) pour allumer des LEDs.
De même, il est possible d'envoyer un signal au microcontrôleur, depuis un capteur par exemple. Ce signal est un courant électrique entrant dans la broche. En fonction du signal reçu, le microcontrôleur effectuera une tâche prévue (allumer la lumière lorsqu'un capteur de mouvement détecte une présence, par exemple). Pour cela, nous utiliserons les commandes DigitalRead et AnalogRead.

Protéger l'Arduino

Jusqu'à maintenant, nous nous sommes toujours contentés de faire circuler du courant du microcontrôleur à la masse.
Au lieu d'utiliser les broches du microcontrôleur seulement sous forme de **sortie** (*OUTPUT*), nous allons aussi les utiliser sous forme d'**entrée** (*INPUT*); c'est-à-dire qu'au lieu d'être raccordée à la masse, la broche le sera au +5V. Tout se passera bien, si au niveau du programme, la broche est configurée comme *input*. Mais si elle devait être configurée en *output* par erreur, il est presque certain que le microcontrôleur finira immédiatement au paradis des puces électroniques grillées.

Ainsi, pour éviter de condamner notre microcontrôleur à la chaise électrique, nous allons devoir le protéger. Cela peut se faire en connectant sur la broche du microcontrôleur utilisé comme *input* une résistance de 100 à 200Ω.



ATTENTION!

Lorsque vous branchez l'Arduino à l'ordinateur, le dernier programme reçu est exécuté. Avant de commencer un nouveau montage, il est plus que conseillé d'envoyer un programme d'initialisation, du modèle de celui-ci:

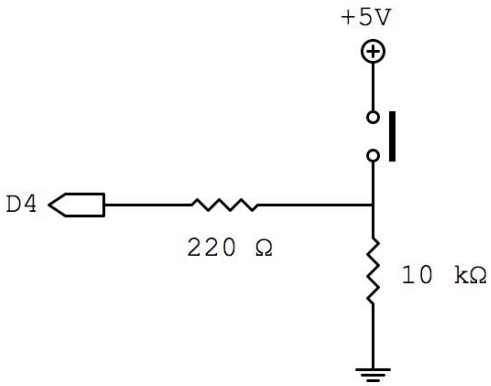
```
void setup() {
}

void loop() {
}
```

Maintenant que ce point est réglé, voyons comment utiliser le bouton poussoir avec le microcontrôleur.

Résistance Pull-Down / Pull-Up

Circuit avec une une résistance pull-down

Observons ce schéma:



Lorsque l'on presse le bouton poussoir, on envoie un courant électrique sur la broche 4, qui sera interprété comme un **1**.
Lorsqu'on relâche le bouton, il n'y a plus de courant qui arrive sur la broche 4, ce qui sera interprété comme un **0**. On a donc un signal binaire: allumé/éteint (on/off).

Ce montage contient une résistance de 10 kΩ (soit 10’000 Ω), qu’on appelle pull-down (littéralement *tirer-en-bas*). Cette résistance permet de *tirer* le potentiel vers le *bas (pull-down).* En français, on appelle aussi ceci un *rappel au moins*.

En effet, contrairement au cas théorique, fermer ou ouvrir un circuit (via un interrupteur ou un bouton poussoir) ne génère pas forcément un signal clair:

- Le circuit non-connecté à la source peut agir comme une antenne dans un environnement pollué d'ondes électromagnétiques (proches d'un téléphone cellulaire, par exemple). Cela va générer dans le circuit un courant qui peut être interprété comme un signal. On appelle ce phénomène *induction*. C'est ainsi, par exemple, que certains smartphones peuvent se recharger sans fil.
- D'un point de vue mécanique, lorsqu'on appuie sur un bouton-poussoir, le contact n'est jamais instantané ni parfait. Il y a des rebonds. Il en est de même lorsqu'on le relâche. Ces phénomènes sont des parasites qui peuvent induire l'Arduino en erreur.

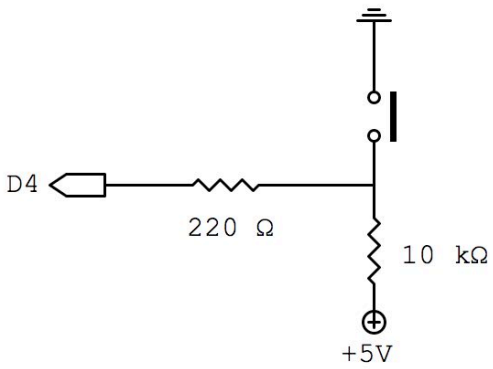
Le but d'une résistance de pull-down est donc d’évacuer les courants vagabonds et de donner un signal clair.
Si on presse le bouton, un courant électrique clair est alors appliqués sur l'entrée. Le courant va prendre le chemin le plus simple, soit par la résistance de 220 Ω et finit par arriver sur D4, qui est relié à la terre. Si on relâche le bouton, la résistance pull-down ramène l'entrée à la terre. Comme D4 est aussi relié à la même terre, il y a alors une différence de potentiel (une tension) de 0 Volts, et donc pas de signal parasite à l'entrée de D4.

Circuit avec une résistance pull-up

Observons maintenant ce schéma à droite:

Si on le compare au schéma d’un circuit avec une résistance pull-down, on constate une inversion entre la terre et le +5V.

En effet, lorsque le circuit est ouvert, une tension de +5V est appliquée à l’entrée de la broche 4.
Lorsqu’on appuie sur le bouton poussoir, le courant électrique va passer par le chemin offrant le moins de résistance, soit directement par la masse (Gnd), sans passer par l’entrée de la broche 4.
Le fonctionnement est donc inversé par rapport à la résistance pull-down.



Résistance pull-down ou pull-up?

A notre niveau, la seule chose qui va changer entre une résistance pull-down et une résistance pull-up est la lecture de l’information:

Avec une résistance pull-down, par défaut, l’entrée sur la broche est égale à 0.

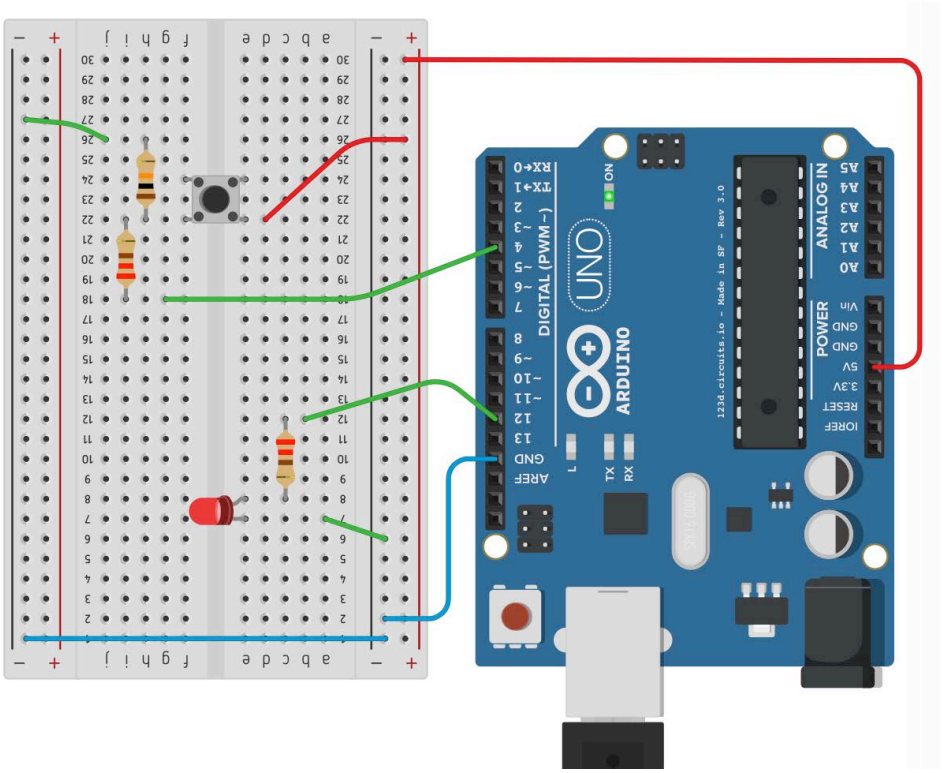
Avec une résistance pull-up, par défaut, l’entrée sur la broche est égale à 1.

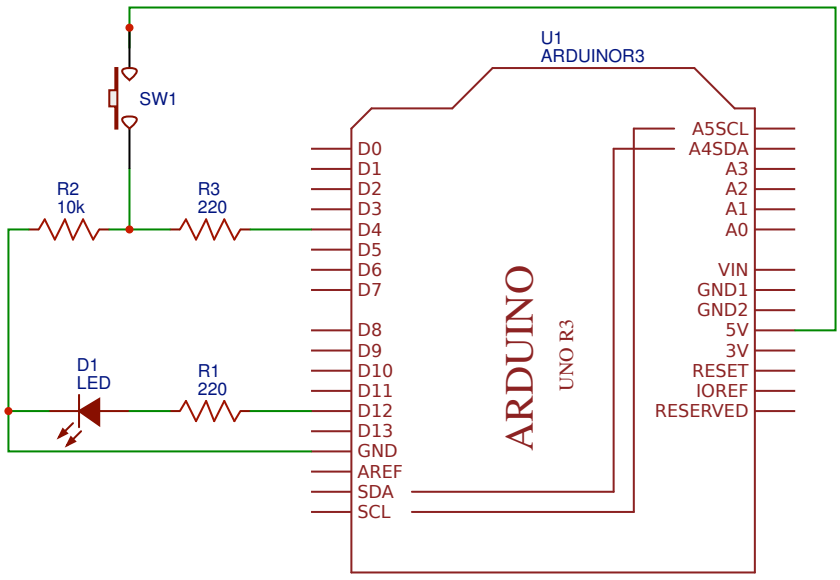
Dans le code `if (digitalRead(bouton) == 1 )` , sera respectivement la valeur d’entrée lorsque le circuit est fermé (pull-down) ou ouvert (pull-up).

Circuit 6: montage avec résistance pull-down (rappel au moins)

Voici le circuit à réaliser:

Circuit 6





Liste des composants:

- 1 Led rouge
- 2 résistances de 220 à 470Ω
- 1 résistance de 1k à 10kΩ
- 1 bouton-poussoir

Lorsqu'on appuie sur le bouton poussoir, La LED doit s'allumer. Et naturellement, lorsqu'on relâche le bouton poussoir, la LED s'éteint. Cette action n'est pas mécanique, mais logique. Ce n'est pas la fermeture d'un circuit électrique qui allume la LED, mais l'information transmise à l'Arduino, par le biais d'un *INPUT* qui lui ordonne d'allumer la LED.

Observons maintenant le code:

Code 10: allumer une LED en fonction de l'état du bouton poussoir

```
/*
Code 10 - Edurobot.ch, destiné à l'Arduino
Allume LED en fonction de l'état du bouton poussoir
*/


// On déclare les variables


const int bouton = 4;          // la broche 4 devient bouton
const int led = 12;            // la broche 12 devient led

void setup()

{
  pinMode(bouton, INPUT);      // Initialise la broche 4 comme entrée
  pinMode(led, OUTPUT); // Initialise la broche 12 comme sortie
  Serial.begin(9600);          // Ouvre le port série à 9600 bauds
}

void loop()

{

// Si bouton poussoir appuyé...

  if (digitalRead(bouton) == 1 )      //teste si le bouton a une valeur de 1

 //...on allume la LED
  {
  digitalWrite(led, HIGH);   // allume la LED
  }

// Sinon...

  else
                        //teste si le bouton a une valeur de 0
//...on éteint la LED
  {
  digitalWrite(led, LOW);   // éteint la LED
  }
}
```

Analysons le code:

Nous utilisons une nouvelle instruction: *if ... else* (*si ... sinon*). C'est donc une *condition*.

Voici ce qu'il va se passer:

Si (*if*)le bouton poussoir est pressé (*digitalRead(bouton) == 1*), allumer la LED (*digitalWrite(led, HIGH)*), sinon (*else*) éteindre la LED (*digitalWrite(led, LOW)*).

L'instruction == vérifie si deux expressions sont égales. Si elles le sont, alors le résultat sera vrai (*true*) sinon le résultat sera faux (*false*).

Ainsi:

```
const int bouton = 4;          // la broche 4 devient bouton
const int led = 12;            // la broche 12 devient led
```

On renomme les broches 4 et 12 respectivement *bouton* et *led.* Cela facilitera la lecture du code.

```
void setup() {
  // Initialise la broche 4 comme entrée
  pinMode(bouton, INPUT);

  // Initialise la broche 12 comme sortie
  pinMode(led, OUTPUT);

  // Ouvre le port série à 9600 bauds
  Serial.begin(9600);
}
```

Voilà, maintenant notre microcontrôleur sait qu'il y a quelque chose de connecté sur sa broche 4 et qu'elle est configurée en entrée (*INPUT*). De même, la broche 12 est configurée en sortie (*OUTPUT*).
Maintenant que le bouton est paramétré, nous allons chercher à savoir quel est son état (appuyé ou relâché).
Si le microcontrôleur détecte un courant électrique à sa broche 4, alors il stockera une valeur de *1*.
Dans le cas contraire (différence de potentiel de 0V), il stockera une valeur de *0*.

Pour lire l'état de la broche 4, nous allons utiliser l'expression *digitalRead*. Ainsi:

```
if (digitalRead(bouton) == 1 )
digitalWrite(led, HIGH);   // allume la LED
```

doit se comprendre comme:

```
si la broche 4 a une valeur égale à 1
alors la broche 12 est en position haute = LED allumée
```

et les lignes:

```
else {
digitalWrite(led, LOW);   // éteint la LED
```

doivent se comprendre comme:

```
sinon (c'est à dire: la broche 4 a une valeur égale à 0)
la broche 12 est en position basse = LED éteinte
```

Voici une petite vidéo qui présente l'activité: http://www.scolcast.ch/podcast/61/53-3071

Code 11: Un code plus élégant

Le code précédent est simple, mais manque un peu de finesse; un peu comme un barbare avec une hache à deux mains dans une réception de Monsieur l'Ambassadeur. Voici donc une autre solution, habillée en smoking:

```
/*
Code 11 - Edurobot.ch, destiné à l'Arduino
Allume LED en fonction de l'état du bouton poussoir
*/

const int bouton = 4;          // la broche 4 devient bouton
const int led = 12;            // la broche 12 devient led
int etatbouton;         // variable qui enregistre l'état du bouton


void setup()
{
  pinMode(bouton, INPUT);      // Initialise le bouton comme entrée
  pinMode(led, OUTPUT);        // Initialise la led comme sortie
  etatbouton = LOW;            // Initialise l'état du bouton comme relâché
  Serial.begin(9600);          // Ouvre le port série à 9600 bauds
}

void loop()
{
  etatbouton = digitalRead(bouton); //On mémorise l'état du bouton

  if(etatbouton == LOW) //teste si le bouton a un niveau logique BAS
   {
       digitalWrite(led,LOW); //la LED reste éteinte
   }

  else  //teste si le bouton a un niveau logique différent de BAS (donc HAUT)
   {
       digitalWrite(led,HIGH); //le bouton est appuyé, la LED est allumée
   }
}
```

Pour commencer, nous allons créer une variable que nous choisirons d'appeler *etatbouton.* Elle servira à stocker l'état du bouton (logique, non?):

```
int etatbouton;
```

On inscrit l'état du bouton dans la variable de cette manière, avec la fonction *digitalRead*:

```
etatbouton = digitalRead(bouton);
```

Il ne nous reste plus qu'à appeler l'état du bouton, stocké dans la variable *etatbouton*, et à lui faire passer un petit test avec *if... else* (si... sinon):

Si (if) l'état du bouton est *LOW* (c'est-à-dire = 0), alors la LED est *LOW* (éteinte). Sinon (else), la LED est *HIGH* (en effet, si l'état du bouton n'est pas *LOW*, il ne peut être que *HIGH*, soit allumée...)

Et cela donne donc ceci:

```
if(etatbouton == LOW) //teste si le bouton a un niveau logique BAS
  {
      digitalWrite(led,LOW); //la LED reste éteinte
  }
 else  //teste si le bouton a un niveau logique différent de BAS (donc HAUT)
  {
      digitalWrite(led,HIGH); //le bouton est appuyé, la LED est allumée
  }
```

Rappelons que l'instruction == vérifie si deux expressions sont égales.

Petits exercices: bouton poussoir et LED qui clignote

Imaginons maintenant le cas de figure suivant: avec le même circuit: lorsque nous appuyons sur le bouton, la LED commence à clignoter.

Modifie le programme pour arriver à ce résultat.

Télécharger la réponse

http://arduino.education/codes/code11-2.zip

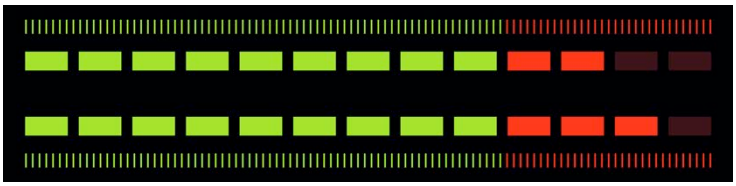
Et maintenant, modifie le programme pour que lorsque nous branchons l'Arduino à l'ordinateur la LED s'allume, et reste allumée. Par contre, quand nous appuyons sur le bouton, la LED clignote.

Télécharger la réponse

http://arduino.education/codes/code11-3.zip

Le bargraphe

Un bargraphe est un afficheur qui indique une quantité, provenant d'une information quelconque (niveau d'eau, puissance sonore, etc.), sous une forme lumineuse. Le plus souvent, on utilise des LEDs alignées en guise d'affichage.



L'objectif de cet exercice est de réaliser un bargraphe de 4 LEDs, avec deux boutons poussoirs. L'un d'eux servira à incrémenter la valeur sur le bargraphe (à savoir augmenter le nombre de LEDs allumées), alors que l'autre servira à le décrémenter.

Voici le schéma du circuit à monter:

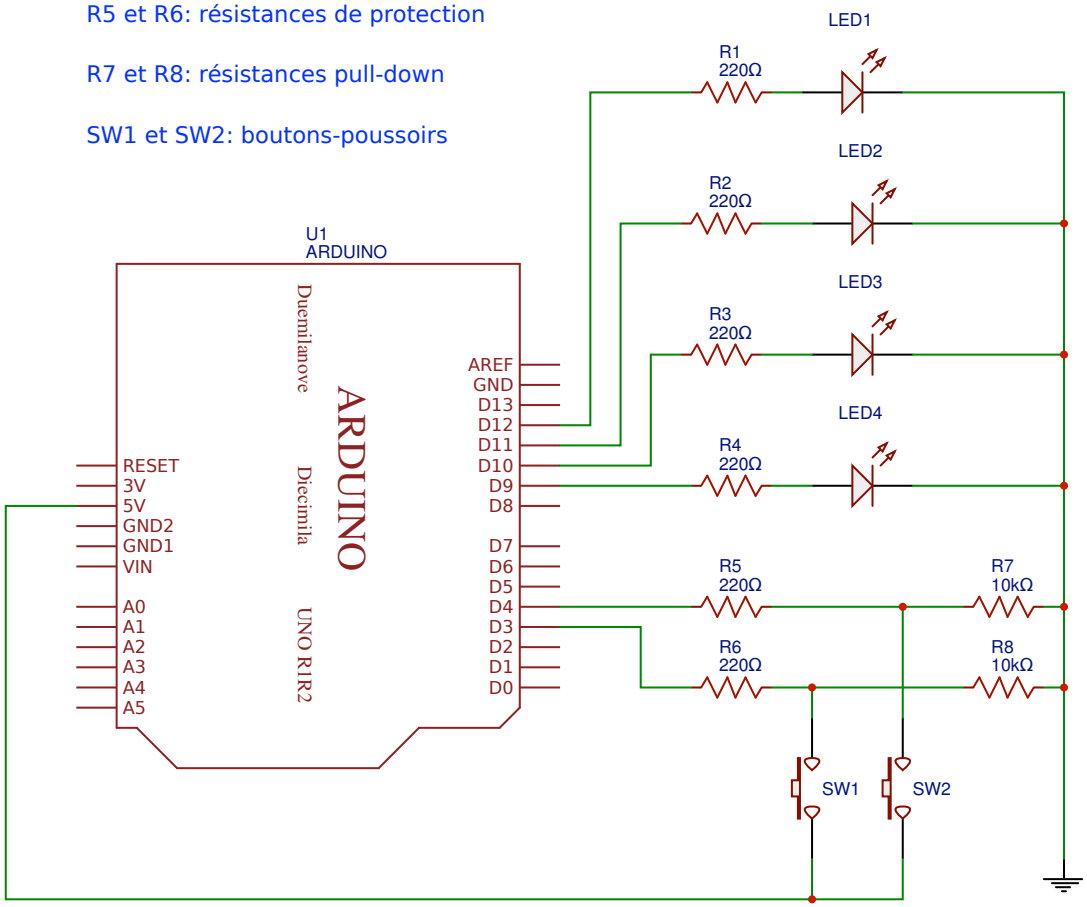
Circuit 7

LED1 à LED4: à choix

R5 et R6: résistances de protection

R7 et R8: résistances pull-down

SW1 et SW2: boutons-poussoirs



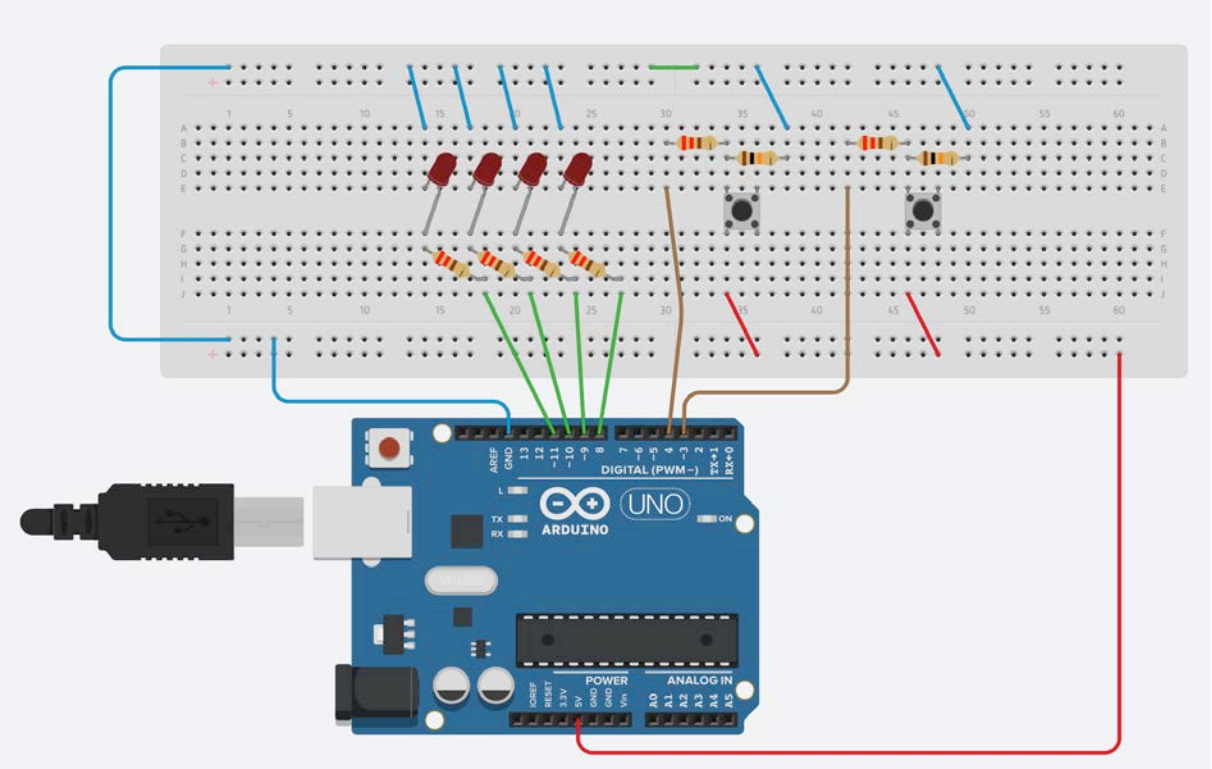
Liste des composants

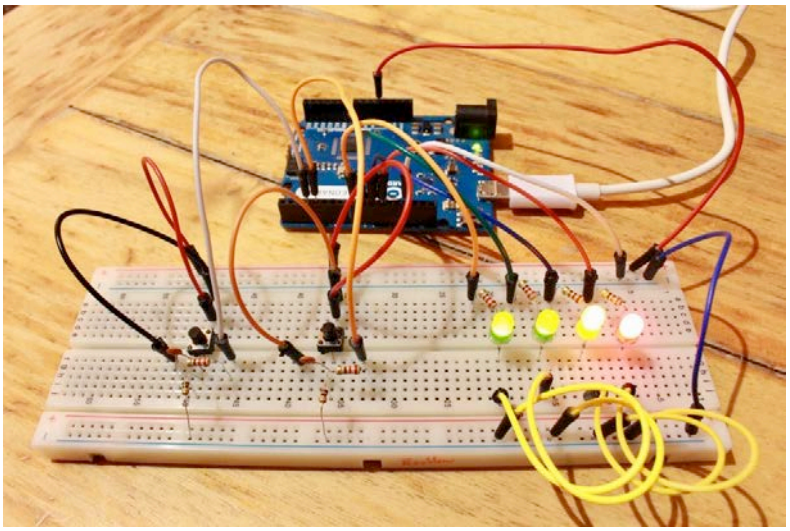
- 4 LEDs à choix
- 6 résistances de 220Ω
- 2 résistances de 1 à 10kΩ
- 2 boutons-poussoirs

Comme on peut l'observer, les résistances R7 et R8 sont montées en pull-down (rappel au moins). Ce circuit pourrait tout à fait être monté avec des résistances pull-up.

Normalement, à ce stade, nous devrions être capables d'exécuter le montage à l'aide du schéma électronique! Dans le cas contraire, voici le montage à réaliser:

Circuit 8





Code 12: le bargraphe

```
/*
Code 12 - Edurobot.ch, destiné à l'Arduino
Le bargraphe
L'objectif est de réaliser un bargraphe, avec 4 LEDs et deux boutons poussoirs: un pour
incrémenter le nombre de LEDs allumées, l'autre pour le décrémenter.
 */


/*
Déclaration des constantes pour les noms des broches
*/

const int btn_minus = 3;   //Bouton 1 pour décrémenter le nombre de LEDs allumées
const int btn_plus = 4;    //Bouton 2 pour incrémenter le nombre de LEDs allumés
const int led_0 = 8;       //Led 0
const int led_1 = 9;       //Led 1
const int led_2 = 10;      //Led 2
const int led_3 = 11;      //Led 3


/*
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; en effet, il n'est pas prévu d'appuyer sur les deux boutons simultanément)
int memoire_plus = LOW;   //état relâché par défaut pour le bouton 2
int memoire_minus = LOW;  //état relâché par défaut pour le bouton 1


/*
Initilisation des broches en entrée/sortie: entrées pour les boutons, sorties pour les LEDs
*/

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);
}


/*
Et c'est parti pour le programme!
*/

void loop()
{
   //lecture de l'état du bouton d'incrémentation (on lit l'état du btn_plus et on l'inscrit
dans la variable etat_bouton)
   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 == HIGH))
   {
      nombre_led++; //on incrémente la variable qui indique combien de LEDs devront 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 == HIGH))
   {
      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 (puisqu'on a 4 LEDs)
   if(nombre_led > 4)
   {
      nombre_led = 4;
   }
   if(nombre_led < 0)
   {
      nombre_led = 0;
   }


   //On crée une fonction affiche() pour l'affichage du résultat
   //on lui envoie alors 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, LOW);
    digitalWrite(led_1, LOW);
    digitalWrite(led_2, LOW);
    digitalWrite(led_3, LOW);

    //Puis on les allume une à une

    if(valeur_recue >= 1)            // "si la valeur reçue est plus grande ou égale à 1..."
    {
        digitalWrite(led_0, HIGH);    // "on allume la LED 0
    }
    if(valeur_recue >= 2)            // "si la valeur reçue est plus grande ou égale à 2..."
    {
        digitalWrite(led_1, HIGH);    // "on allume la LED 1 (sous-entendu que la LED 0 est
allumée, puisque la valeur est plus grande que 1)
    }
    if(valeur_recue >= 3)            // "si la valeur reçue est plus grande ou égale à 3..."
    {
        digitalWrite(led_2, HIGH);    // "on allume la LED 2
    }
    if(valeur_recue >= 4)            // "si la valeur reçue est plus grande ou égale à 4..."
    {
        digitalWrite(led_3, HIGH);    // "on allume la LED 3
    }
}
```

Vidéo

Vidéo de présentation du bargraphe: http://www.scolcast.ch/episode/arduino-lecole-le- -1

Analyse du code:

En se référant au schéma de montage, on déclare les constantes pour chaque broche: les boutons poussoirs sont connectés sur les broches 3 et 4, alors que les LEDs sont connectées aux broches 8 à 11.

```
/*
Déclaration des constantes pour les noms des broches
*/

const int btn_minus = 3;   //Bouton 1 pour décrémenter le nombre de LEDs allumées
const int btn_plus = 4;    //Bouton 2 pour incrémenter le nombre de LEDs allumés
const int led_0 = 8;       //Led 0
const int led_1 = 9;       //Led 1
const int led_2 = 10;      //Led 2
const int led_3 = 11;      //Led 3
```

On crée ensuite les variables nécessaires et on initialise leur état.

```
/*
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; en effet, il n'est pas prévu d'appuyer sur les deux boutons simultanément)
int memoire_plus = LOW;   //état relâché par défaut pour le bouton 2
int memoire_minus = LOW;  //état relâché par défaut pour le bouton 1
```

On initialise ensuite les broches, selon qu'il s'agit des entrées (les boutons) ou des sorties (les LEDs).

```
/*
Initialisation des broches en entrée/sortie: entrées pour les boutons, sorties pour les LEDs
*/

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);
}
```

Allons-y pour la partie du programme dans la boucle sans fin (loop).

On utilise digitalRead pour lire l'état du bouton (btn_plus) et inscrire cet état dans la variable etat_bouton.

```
void loop()
{
   //lecture de l'état du bouton d'incrémentation (on lit l'état du btn_plus et on l'inscrit
dans la variable etat_bouton)

   etat_bouton = digitalRead(btn_plus);
```

Maintenant, ça devient sérieux. Regardons ce qui suit:

```
  //Si le bouton a un état différent que celui enregistré ET que cet état est "appuyé"

   if((etat_bouton != memoire_plus) && (etat_bouton == HIGH))
```

Traduisons cela en français: *si l'état du bouton 2 est différent de celui enregistré, et que quelqu'un appuie sur le bouton, alors...*

Le != signifie est différent de (teste la différence entre deux variables) et l'opérateur logique && signifie ET (Pour être précis, nous avons: *si ... et...* avec le *if ... &&...* Exemple: *si* il fait beau *et* chaud, alors on va à la plage).

Rappelons-nous maintenant de l'opération ++, qui incrémente une variable (variable = variable + 1, c'est-à-dire; on ajoute à chaque fois 1 à la variable). Dans notre cas, il s'agit de la valeur de la variable nombre_led qu'on incrémente.

```
   {
      nombre_led++; //on incrémente la variable qui indique combien de LEDs devront s'allumer
   }
```

Et zou! On enregistre le nouvel état du bouton pour la suite!

```
   memoire_plus = etat_bouton; //on enregistre l'état du bouton pour le tour suivant.
```

Maintenant, on recommence le tout, mais pour le bouton 1.

```
   //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 == HIGH))
   {
      nombre_led--; //on décrémente la valeur de nombre_led
   }
   memoire_minus = etat_bouton; //on enregistre l'état du bouton pour le tour suivant
```

Nous allons maintenant limiter le nombre de LEDs à connecter. En effet, dans notre cas, nous avons 4 LEDs. Alors si on appuie 10 fois sur le bouton 2, cela n'allumera que les 4 LEDs.

```
   //on applique des limites au nombre pour ne pas dépasser 4 ou 0 (puisqu'on a 4 LEDs)
   if(nombre_led > 4)
   {
      nombre_led = 4;
   }
```

Traduisons: *si on appuie plus de 4 fois sur le bouton, le résultat sera égal à 4*

Même chose maintenant pour le bouton 1.

```
   if(nombre_led < 0)
   {
      nombre_led = 0;
   }
```

Maintenant, nous devons gérer l'allumage des LEDs. Pour simplifier le code, on va créer une fonction qui servira à gérer l'affichage. Nous allons appeler cette fonction affiche, avec un paramètre int valeur_recue. Ce paramètre représente le nombre à afficher.

```
   //On créé une fonction affiche() pour l'affichage du résultat
   //on lui envoie alors en paramètre la valeur du nombre de LED à éclairer

   affiche(nombre_led);
}

void affiche(int valeur_recue)
```

On commence d'abord par éteindre toutes les LEDs.

```
{
    //on éteint toutes les leds
    digitalWrite(led_0, LOW);
    digitalWrite(led_1, LOW);
    digitalWrite(led_2, LOW);
    digitalWrite(led_3, LOW);
```

Si la fonction reçoit le nombre 1, on allume la LED 1. Si elle reçoit le nombre 2, elle allume la LED 1 et 2. Si elle reçoit 3, elle allume la LED 1, 2 et 3. Enfin, si elle reçoit 4, alors elle allume toutes les LEDs:

```
    //Puis on les allume une à une

    if(valeur_recue >= 1)            // "si la valeur reçue est plus grande ou égale à 1..."
    {
        digitalWrite(led_0, HIGH);    // "on allume la LED 0
    }
    if(valeur_recue >= 2)            // "si la valeur reçue est plus grande ou égale à 2..."
    {
        digitalWrite(led_1, HIGH);    // "on allume la LED 1 (sous-entendu que la LED 0 est
allumée, puisque la valeur est plus grande que 1)
    }
    if(valeur_recue >= 3)            // "si la valeur reçue est plus grande ou égale à 3..."
    {
        digitalWrite(led_2, HIGH);    // "on allume la LED 2
    }
    if(valeur_recue >= 4)            // "si la valeur reçue est plus grande ou égale à 4..."
    {
        digitalWrite(led_3, HIGH;    // "on allume la LED 3
    }
}
```

Le symbole >= signife: ...est supérieur ou égal à.... Il s'agit de tester la supériorité ou l'égalité d'une variable par rapport à une valeur. Ainsi, dans:

```
if(valeur_recue >= 4)
   {
        digitalWrite(led_3, HIGH);    // "on allume la LED 3
    }
```

Il faut lire: *si la variable valeur_recue est supérieure ou égale à 4, alors on allume la LED 3*.

Déparasiter à l'aide de condensateurs

Vous l'avez sans doute constaté: malgré des pull-down, les boutons-poussoirs sont peu précis. En effet, parfois, deux LEDs s'allument ou s'éteignes pour une pression sur le bouton. Cela est dû au fait que le bouton-poussoir n'est mécaniquement pas parfait. Lorsqu'on appuie dessus, le signal n'est pas forcément propre. Pendant quelques millisecondes, le signal va passer de 0V à 5V plusieurs fois avant de se stabiliser. L'Arduino peut interpréter un de ces mouvements parasites pour un signal d'entrée et va donc réagir en fonction. Par exemple, en appuyant une fois sur le bouton-poussoir, l'Arduino peut enregistrer deux impulsions en entrée et allumera deux LEDs au lieu d'une seule.

Il y a moyen de déparasiter le bouton-poussoir et d'absorber ces rebonds en montant en parallèle du bouton-poussoir un condensateur de faible capacité (10nF).

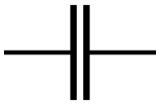
Qu'est-ce qu'un condensateur?

Le condensateur est un composant électronique passif, comme les résistances. Il a pour faculté d'emmagasiner une charge électrique, avant de pouvoir la restituer en cas de baisse de la tension. On utilise ainsi les condensateur comme régulateur de tension, Mais on utilise aussi sa capacité à se charger pour absorber les brusques, mais courtes fluctuations de tension que sont les parasites. La capacité de charge se mesure en Farads (*F*).

Condensateur céramique

Condensateur électrolytique

Symbole du condensateur



Circuit 9

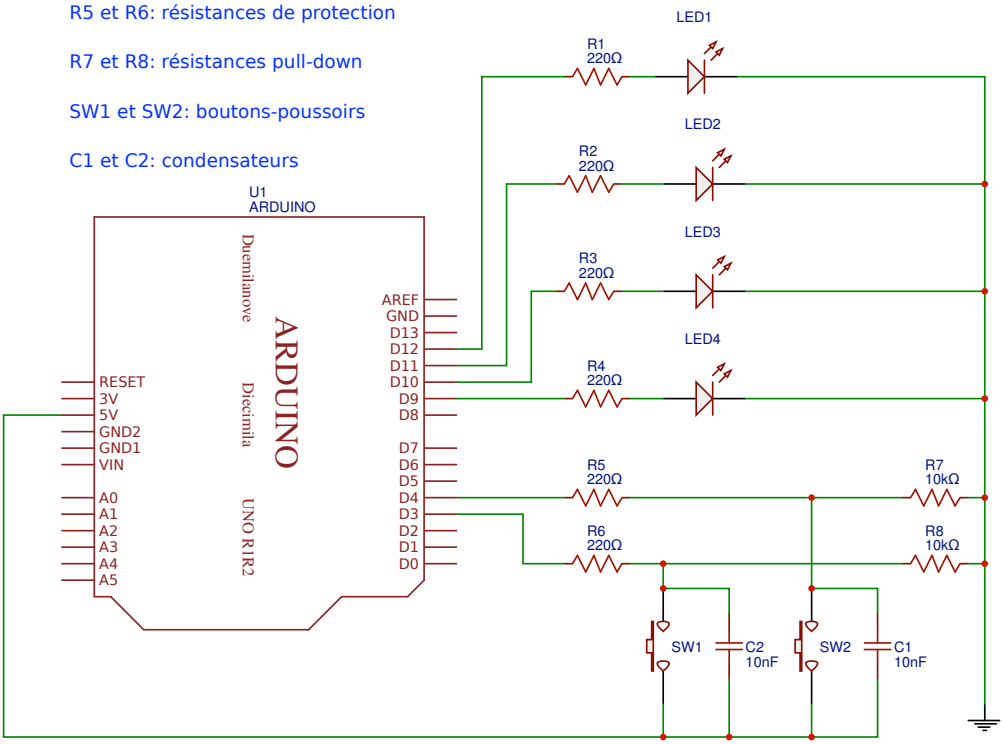
LED1 à LED4: à choix

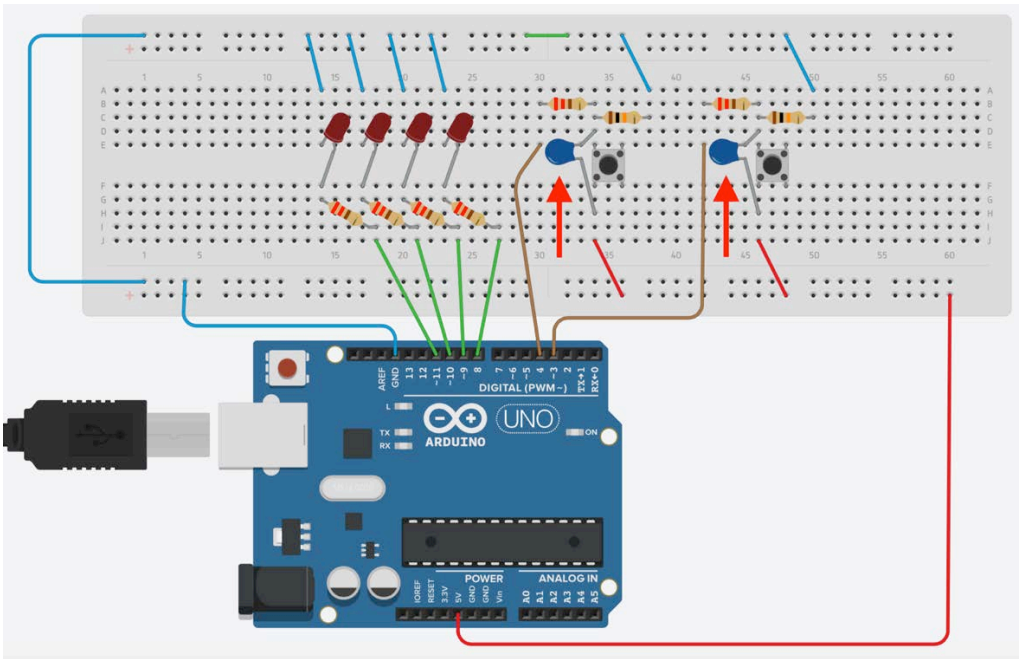
R5 et R6: résistances de protection

R7 et R8: résistances pull-down

SW1 et SW2: boutons-poussoirs

C1 et C2: condensateurs



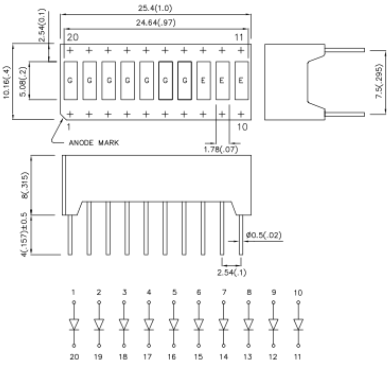


Comme on peut le voir sur le circuit ci-dessus, deux condensateurs de faible capacité ont été ajoutés en parallèle aux boutons-poussoirs. Si la parade n'est pas absolue, cela permet de considérablement augmenter la précision des boutons.

L'installation des condensateurs n'a aucune influence au niveau du code. Il n'est donc pas nécessaire de le modifier.

Variation: Le bargraphe à 10 LEDs

Il existe des bargraphes intégrants 10 LEDs. On peut les utiliser en remplacement des 4 LEDs du précédent montage.

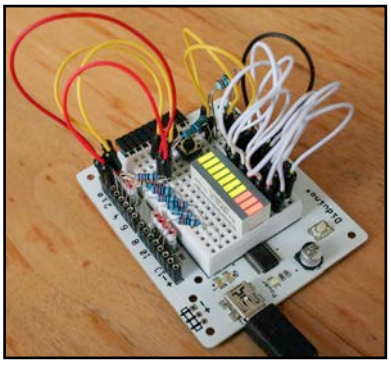


Références utilisées dans cet exercice: Kingbright DC-10EWA ou DC7G3EWA.

D'un côté se trouvent les cathodes, de l'autre les anodes. Dans notre cas, les anodes sont du côté où se trouvent les inscriptions.

Le code, lui, doit être adapté en conséquence pour 10 LEDs. Un exemple se trouve ici:

Code 13: le bargraphe à 10 LEDs

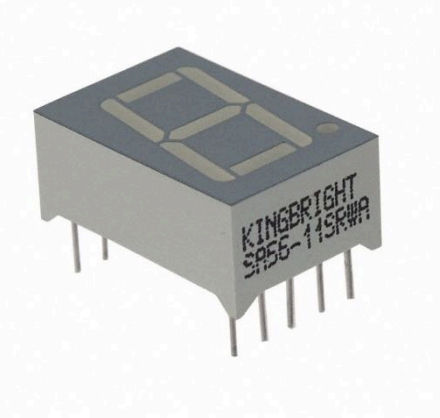


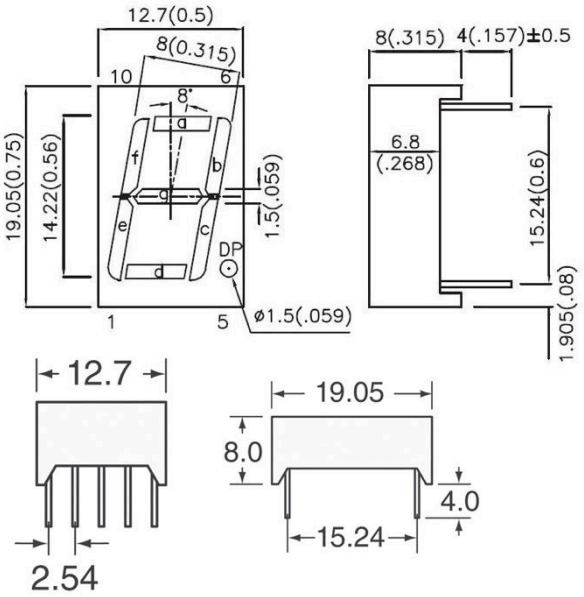
```
/*
  Code 13 - Edurobot.ch, destiné à l'Arduino
  Le barregraphe
  L'objectif est de réalier un barregraphe, avec deux boutons
poussoirs: un pour incrémenter le nombre de LEDs allumées,
l'autre pour le décrémenter.
  Le barregraphe est composé de 10 LEDs.
*/


/*
  Déclaration des constantes pour les noms des broches
*/

const int btn_minus = 2;   //Bouton 1 pour décrémenter le nombre de LEDs allumés
const int btn_plus = 3;    //Bouton 2 pour incrémenter le nombre de LEDs allumés
const int led_0 = 4;       //Led 0
const int led_1 = 5;       //Led 1
const int led_2 = 6;      //Led 2
const int led_3 = 7;      //Led 3
const int led_4 = 8;      //Led 4
```

```
const int led_5 = 9;      //Led 5
const int led_6 = 10;      //Led 6
const int led_7 = 11;     //Led 7
const int led_8 = 12;     //Led 8
const int led_9 = 13;     //Led 9

/*
  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; en effet, il n'est pas prévu d'appuyer sur les deux boutons simultanément)
int memoire_plus = LOW;   //état relâché par défaut pour le bouton 2
int memoire_minus = LOW;  //état relâché par défaut pour le bouton 1


/*
  Initialisation des broches en entrée/sortie: entrées pour les boutons, sorties pour les LEDs
*/

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);
  pinMode(led_4, OUTPUT);
  pinMode(led_5, OUTPUT);
  pinMode(led_6, OUTPUT);
  pinMode(led_7, OUTPUT);
  pinMode(led_8, OUTPUT);
  pinMode(led_9, OUTPUT);

}

void loop()
{
  //lecture de l'état du bouton d'incrémentation (on lit l'état du btn_plus et on l'inscrit
 //dans la variable etat_bouton)
  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 == HIGH))
  {
    nombre_led++; //on incrémente la variable qui indique combien de LEDs 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 == HIGH))
  {
    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 (puisqu'on a 4 LEDs)
  if (nombre_led > 10)
  {
    nombre_led = 10;
  }
  if (nombre_led < 0)
  {
```

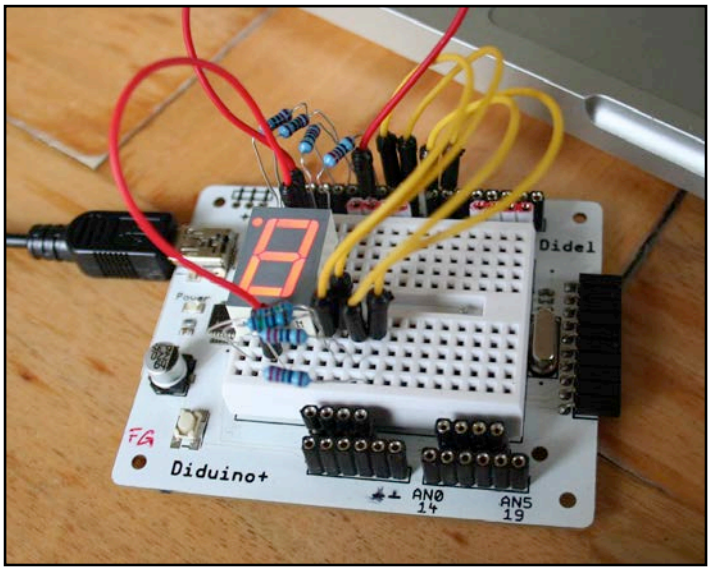
```
    nombre_led = 0;
  }


  //On créé une fonction affiche() pour l'affichage du résultat
  //on lui envoie alors 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, LOW);
  digitalWrite(led_1, LOW);
  digitalWrite(led_2, LOW);
  digitalWrite(led_3, LOW);
  digitalWrite(led_4, LOW);
  digitalWrite(led_5, LOW);
  digitalWrite(led_6, LOW);
  digitalWrite(led_7, LOW);
  digitalWrite(led_8, LOW);
  digitalWrite(led_9, LOW);

  //Puis on les allume une à une

  if (valeur_recue >= 1)           // "si la valeur reçue est plus grande ou égale à 1..."
  {
    digitalWrite(led_0, HIGH);    // "on allume la LED 0
  }
  if (valeur_recue >= 2)           // "si la valeur reçue est plus grande ou égale à 2..."
  {
    digitalWrite(led_1, HIGH);    // "on allume la LED 1 (sous-entendu que la LED 0 est
allumée, puisque la valeur est plus grande que 1)
  }
  if (valeur_recue >= 3)           // "si la valeur reçue est plus grande ou égale à 3..."
  {
    digitalWrite(led_2, HIGH);    // "on allume la LED 2
  }
  if (valeur_recue >= 4)           // "si la valeur reçue est plus grande ou égale à 4..."
  {
    digitalWrite(led_3, HIGH);    // "on allume la LED 3
  }
  if (valeur_recue >= 5)           // "si la valeur reçue est plus grande ou égale à 4..."
  {
    digitalWrite(led_4, HIGH);    // "on allume la LED 3
  }
  if (valeur_recue >= 6)           // "si la valeur reçue est plus grande ou égale à 4..."
  {
    digitalWrite(led_5, HIGH);    // "on allume la LED 3
  }
  if (valeur_recue >= 7)           // "si la valeur reçue est plus grande ou égale à 4..."
  {
    digitalWrite(led_6, HIGH);    // "on allume la LED 3
  }
  if (valeur_recue >= 8)           // "si la valeur reçue est plus grande ou égale à 4..."
  {
    digitalWrite(led_7, HIGH);    // "on allume la LED 3
  }
  if (valeur_recue >= 9)           // "si la valeur reçue est plus grande ou égale à 4..."
  {
    digitalWrite(led_8, HIGH);    // "on allume la LED 3
  }
  if (valeur_recue >= 10)           // "si la valeur reçue est plus grande ou égale à 4..."
  {
    digitalWrite(led_9, HIGH);    // "on allume la LED 3
  }
}
```

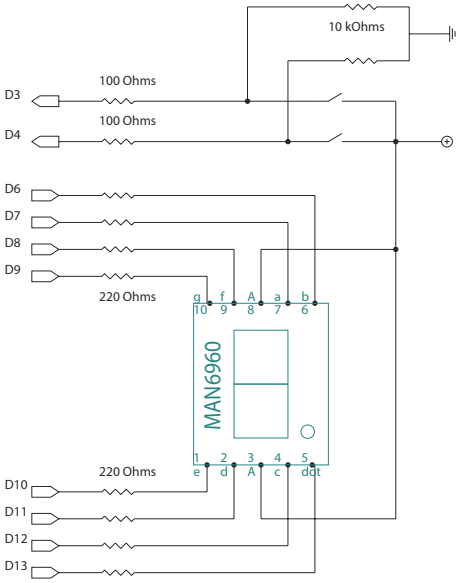
Variation: l'afficheur numérique





Pour ce montage, nous allons utiliser un afficheur numérique à LED.
La référence utilisée ici est un afficheur à anode commune Kingbright SA56-11 SRWA[11].





Dans le cas d'une anode commune, cette dernière est branchée sur le +5V. Les cathodes sont branchées sur les broches.
Au niveau du code, cela implique que les LEDs sont allumée en position LOW et éteintes en position HIGH. Avec un composant à cathode commune, c'est le contraire.
Dans notre exemple, nous allons commencer par allumer toutes les diodes, puis l'une après l'autre, pour les identifier. On peut ensuite écrire des chiffres.

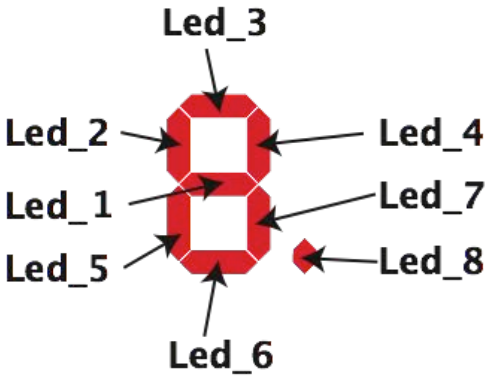
[11] Disponible ici: www.scolcast.ch/episode/arduino-lecole-apprendre-compter

Identification de la position des LEDs

Le code suivant permet d'identifier la position des LEDs en les allumant l'une après l'autre:

```
/*
Code 14 - Edurobot.ch, destiné à l'Arduino
L'afficheur 7 ou 8 segments
L'objectif est d'afficher des chiffres sur un afficheur 7 segments (7 digits).
Ce code a pour objectif d'allumer toutes les LEDS puis l'une après l'autre
pour identifier leur position.
 */

const int led_1 = 6;       //Led 1
const int led_2 = 7;      //Led 2
const int led_3 = 8;      //Led 3
const int led_4 = 9;
const int led_5 = 10;
const int led_6 = 11;
const int led_7 = 12;
const int led_8 = 13;

void setup()
{
   pinMode(led_1, OUTPUT);
   pinMode(led_2, OUTPUT);
   pinMode(led_3, OUTPUT);
   pinMode(led_4, OUTPUT);
   pinMode(led_5, OUTPUT);
   pinMode(led_6, OUTPUT);
   pinMode(led_7, OUTPUT);
   pinMode(led_8, OUTPUT);
}

void loop()
{

    //on allume toutes les leds

    digitalWrite(led_1, LOW);
    digitalWrite(led_2, LOW);
    digitalWrite(led_3, LOW);
    digitalWrite(led_4, LOW);
    digitalWrite(led_5, LOW);
    digitalWrite(led_6, LOW);
    digitalWrite(led_7, LOW);
    digitalWrite(led_8, LOW);
    delay(1500);

    //on éteint toutes les leds

    digitalWrite(led_1, HIGH);
    digitalWrite(led_2, HIGH);
    digitalWrite(led_3, HIGH);
    digitalWrite(led_4, HIGH);
    digitalWrite(led_5, HIGH);
    digitalWrite(led_6, HIGH);
    digitalWrite(led_7, HIGH);
    digitalWrite(led_8, HIGH);
    delay(500);

    //on allume une diode après l'autre pour identifier sa position

    digitalWrite(led_1, LOW);
    delay(1500);

    digitalWrite(led_1, HIGH);
    digitalWrite(led_2, LOW);
    delay(1500);
```

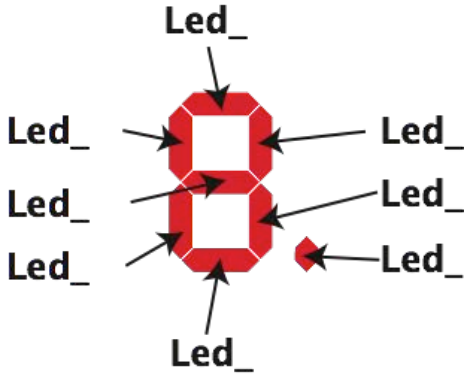
```
    digitalWrite(led_2, HIGH);
    digitalWrite(led_3, LOW);
    delay(1500);

    digitalWrite(led_3, HIGH);
    digitalWrite(led_4, LOW);
    delay(1500);

    digitalWrite(led_4, HIGH);
    digitalWrite(led_5, LOW);
    delay(1500);

    digitalWrite(led_5, HIGH);
    digitalWrite(led_6, LOW);
    delay(1500);

    digitalWrite(led_6, HIGH);
    digitalWrite(led_7, LOW);
    delay(1500);

    digitalWrite(led_7, HIGH);
    digitalWrite(led_8, LOW);
    delay(1500);
}
```



L'objectif est d'identifier chaque LED sur le schéma ci-dessous:
Voici donc, dans notre cas, les LEDs qui doivent être allumées pour écrire les chiffres suivants:

#	led_1	led_2	led_3	led_4	led_5	led_6	led_7
1				X			X
2	X		X	X	X	X	
3	X		X	X		X	X
4	X	X		X			X
5	X	X	X			X	X
6	X	X	X		X	X	X
7			X	X			X
8	X	X	X	X	X	X	X
9	X	X	X	X		X	
0		X	X	X	X	X	X

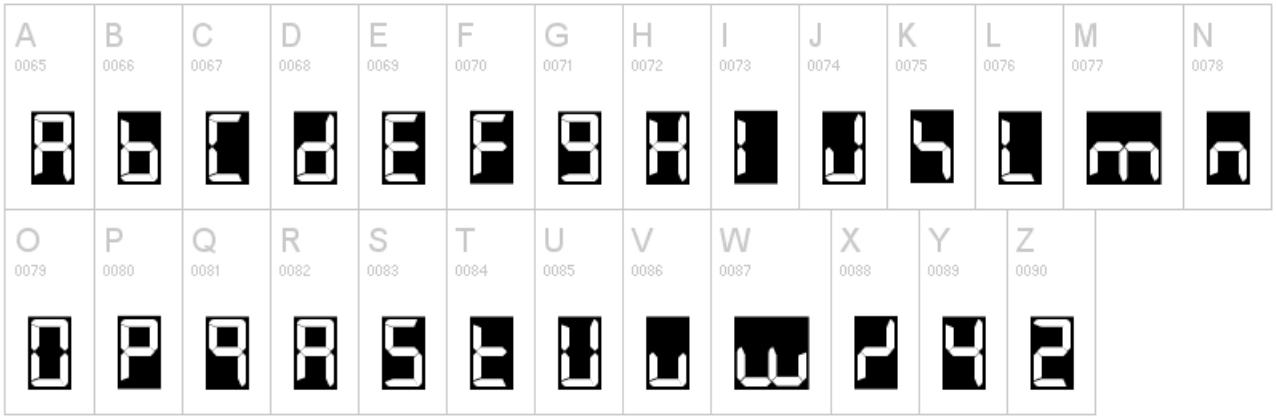
Nous n'utilisons pas la led_8, qui représente un point.
Le code pour compter de 0 à 9 est ici: http://www.edurobot.ch/code/code16.txt



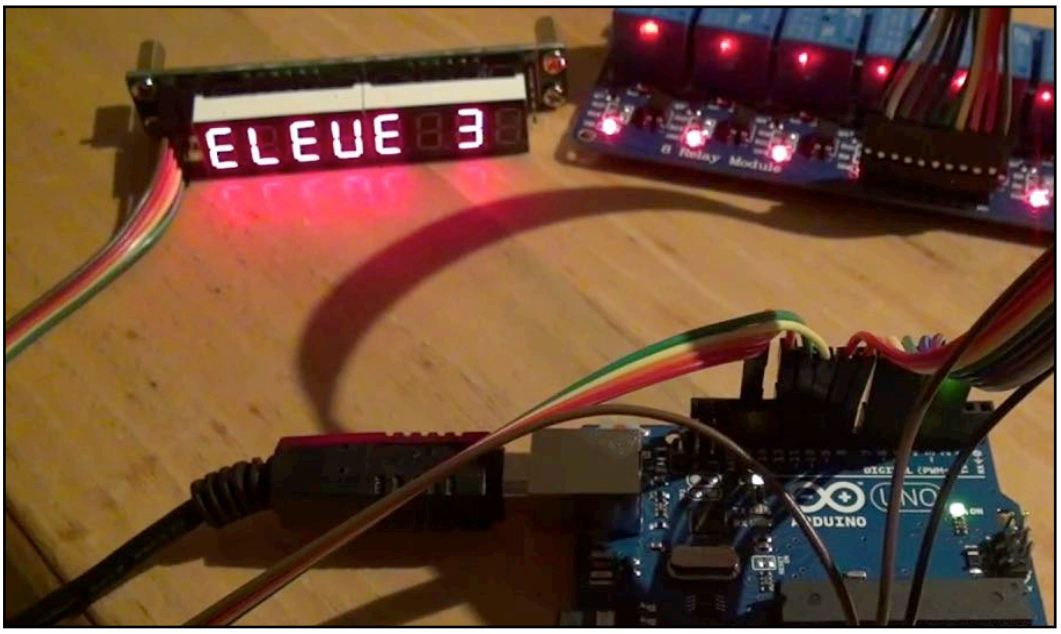
A ton tour:

#	led_1	led_2	led_3	led_4	led_5	led_6	led_7
1							
2							
3							
4							
5							
6							
7							
8							
9							
0							

Il est aussi possible d'écrire les lettres de l'alphabet à l'aide d'un affichage 7 segments[12]:



Exemple:



Évidemment, cela reste artisanal; mais ça peut donner des résultats acceptables, comme on peut le voir ici:

[12] Source de la police: http://www.dafont.com/7led.font

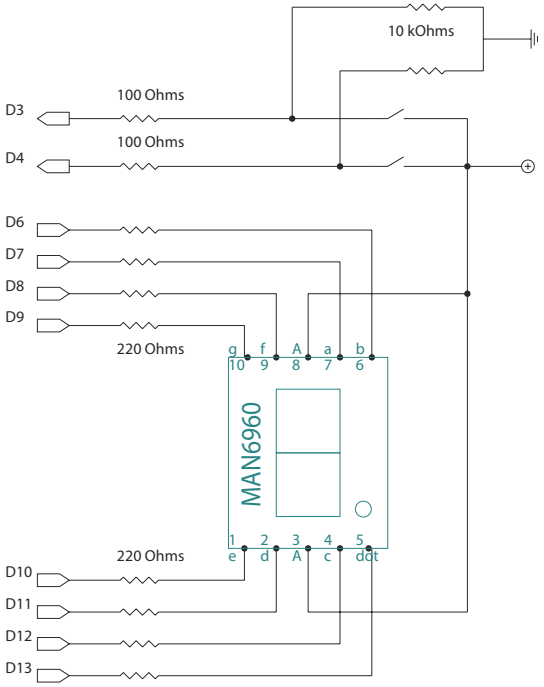
Synthèse: apprendre à compter

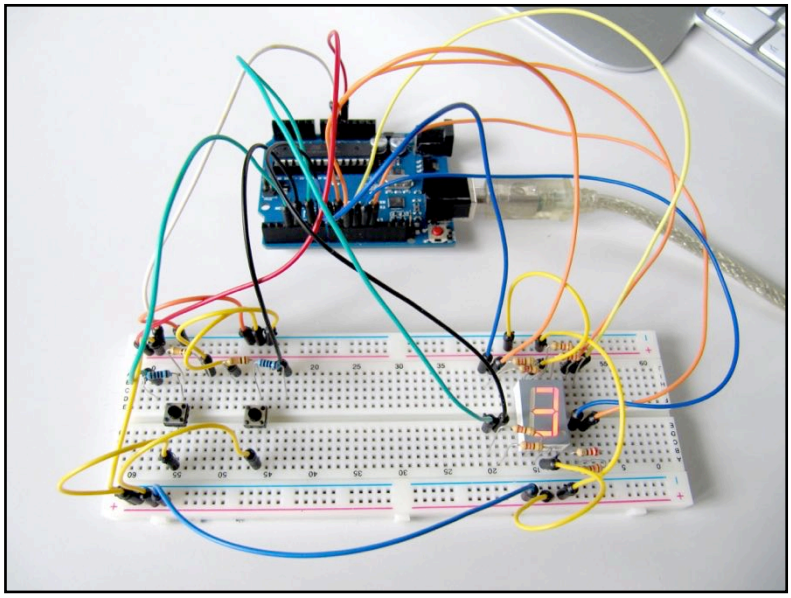
Objectif

Cette dernière leçon va synthétiser tout ce que nous avons vu jusqu'à maintenant. L'objectif est de réaliser un montage, avec deux boutons poussoirs et un affichage 7 segments. L'un des boutons va servir à incrémenter les chiffres sur l'affichage, et l'autre à les décrémenter. Ainsi, en appuyant 6 fois sur le bouton-poussoir, les chiffres de 1 à 6 vont successivement s'afficher.

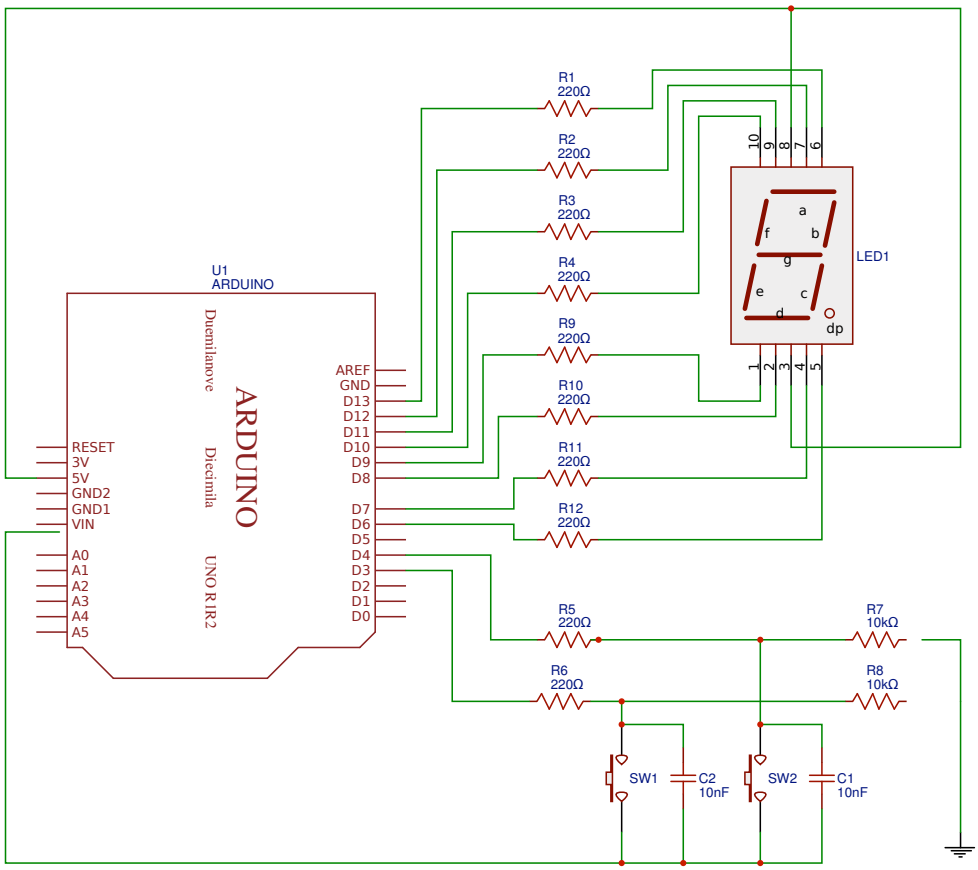
Schéma électronique du montage

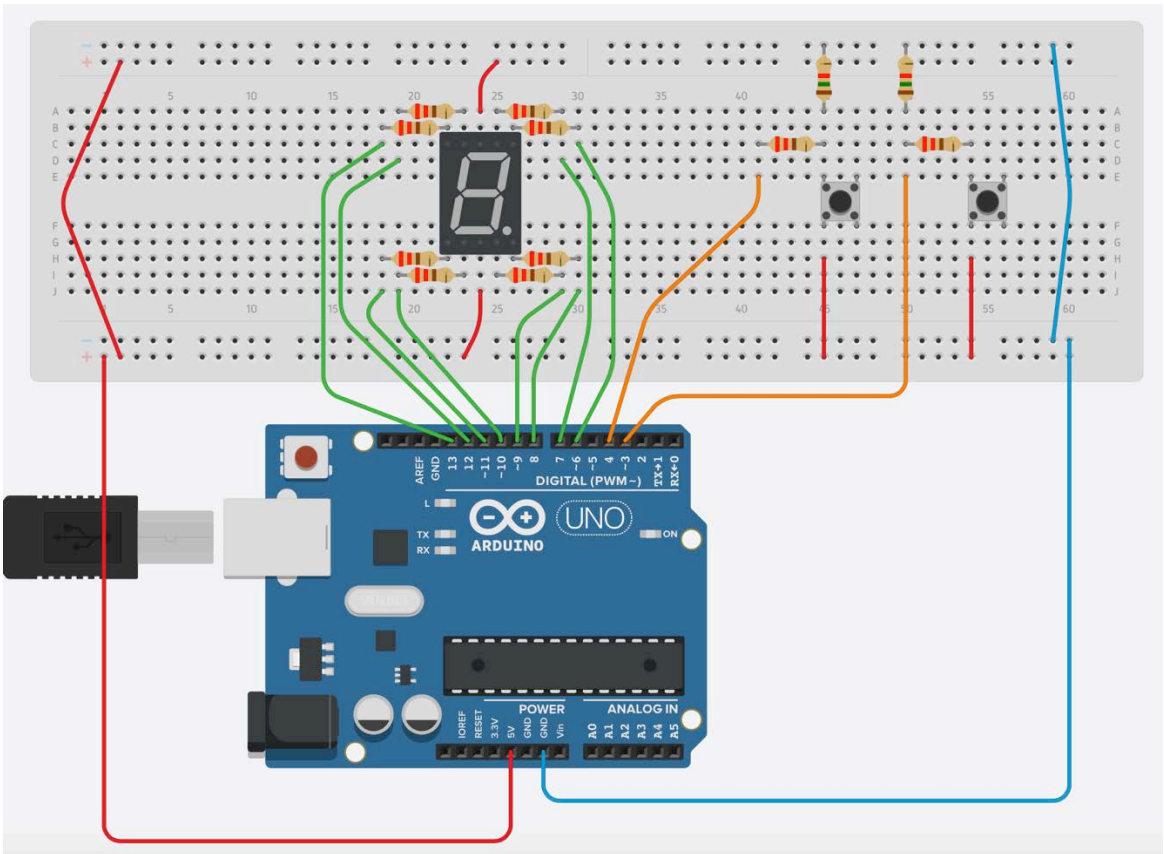
Sur le schéma qui suit, l'affichage 7 segments est à anode commune. Comme d'habitude, chaque bouton-poussoir est doté d'une résistance *pull-down* de 1kΩ ou 10kΩ et d'une résistance de protection de l'*input* de 100Ω. L'ordre de branchement de l'afficheur importe peu, puisqu'on va utiliser le code 14 pour identifier l'ordre d'activation des broches.





Circuit 10


U1
ARDUINO
Duemilanove
ARDUINO
Diecimila
UNO R3 R2
RESET
3V
5V
GND2
GND1
VIN
A0
A1
A2
A3
A4
A5
AREF
GND
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
R1
220Ω
R2
220Ω
R3
220Ω
R4
220Ω
R9
220Ω
R10
220Ω
R11
220Ω
R12
220Ω
R5
220Ω
R6
220Ω
R7
10kΩ
R8
10kΩ
LED1
a
b
c
d
e
f
g
dp
SW1
SW2
C2
10nF
C1
10nF


DIGITAL (PWM~)
UNO
ARDUINO
POWER
ANALOG IN

Code 15: apprendre à compter

Le code peut être téléchargé à l'adresse suivante: http://edurobot.ch/code/code16.txt

Il est très similaire aux codes précédents. La principale différence réside dans le `if(valeur_recue == 1)` , qui dans ce cas signifie *"si la valeur reçue est égale à 1, on allume les segments pour afficher le chiffre 1"*. Contrairement au `if(valeur_recue >= 1)`, qui permet de cumuler l'allumage des LEDs (1 LED pour 1, deux LEDs pour 2,...), l'objectif est ici de n'allumer que les LEDs nécessaires à l'affichage du bon chiffre. Il ne faudrait en effet pas qu'un 3 devienne ensuite un 9 au lieu d'un 4, parce qu'on cumule les LEDs allumées.

```
/*
Code 15 - Edurobot.ch, destiné à l'Arduino
Apprendre à compter
L'objectif est d'afficher des chiffres sur un afficheur 7 segments. Un  bouton permet
d'incrémenter le chiffre, l'autre de le décrémenter.
On compte ainsi de 0 à 9 ou de 1 à 0.
 */


/*
Déclaration des constantes pour les noms des broches.
Notre affichage est un 8 segments, avec donc un point. Même si nous ne l'utilisons pas, nous
l'avons quand même câblé.
*/

const int btn_minus = 3;   //Bouton 1 pour décrémenter les chiffres
const int btn_plus = 4;    //Bouton 2 pour incrémenter les chiffres
const int led_1 = 6;       //Led 1
const int led_2 = 7;       //Led 2
const int led_3 = 8;       //Led 3
const int led_4 = 9;       //Led 4
const int led_5 = 10;
const int led_6 = 11;
const int led_7 = 12;
const int led_8 = 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; en effet, il n'est pas prévu d'appuyer sur les deux boutons simultanément)
int memoire_plus = LOW;   //état relâché par défaut pour le bouton 2
int memoire_minus = LOW;  //état relâché par défaut pour le bouton 1

/*
Inutilisation des broches en entrée/sortie: entrées pour les boutons, sorties pour les LEDs
*/

void setup()
{
  pinMode(btn_plus, INPUT);
  pinMode(btn_minus, INPUT);
  pinMode(led_1, OUTPUT);
  pinMode(led_2, OUTPUT);
  pinMode(led_3, OUTPUT);
  pinMode(led_4, OUTPUT);
  pinMode(led_5, OUTPUT);
  pinMode(led_6, OUTPUT);
  pinMode(led_7, OUTPUT);
  pinMode(led_8, OUTPUT);
}
```

```
/*
Et c'est parti pour le programme!
*/

void loop()
{
  //lecture de l'état du bouton d'incrémentation (on lit l'état du btn_plus et on l'inscrit
  //dans la variable etat_bouton)
  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 == HIGH))
  {
     nombre_led++; //on incrémente la variable qui indique combien de LEDs 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 == HIGH))
  {
     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 (puisqu'on a 4 LEDs)
  if(nombre_led > 10)
  {
     nombre_led = 10;
  }
  if(nombre_led < 0)
  {
     nombre_led = 0;
  }


  //On créé une fonction affiche() pour l'affichage du résultat
  //on lui envoie alors 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_1, HIGH);
   digitalWrite(led_2, HIGH);
   digitalWrite(led_3, HIGH);
   digitalWrite(led_4, HIGH);
   digitalWrite(led_5, HIGH);
   digitalWrite(led_6, HIGH);
   digitalWrite(led_7, HIGH);
   digitalWrite(led_8, HIGH);


//Pour chaque chiffre, le plus simple est de lister toutes les LEDs
//et de leur attribuer la valeur voulue


   if(valeur_recue == 1)            // "si la valeur reçue est égale à 1, on allume les
                                       //segments pour afficher le chiffre 1"
   {

   digitalWrite(led_1, LOW);
   digitalWrite(led_2, HIGH);
```

```
  digitalWrite(led_3, HIGH);
  digitalWrite(led_4, LOW);
  digitalWrite(led_5, HIGH);
  digitalWrite(led_6, HIGH);
  digitalWrite(led_7, HIGH);
  digitalWrite(led_8, HIGH);

  }

  if(valeur_recue == 2)            // "si la valeur reçue est égale à 2, on allume les
                                     //segments pour afficher le chiffre 2"
  {

  digitalWrite(led_1, LOW);
  digitalWrite(led_2, LOW);
  digitalWrite(led_3, LOW);
  digitalWrite(led_4, HIGH);
  digitalWrite(led_5, LOW);
  digitalWrite(led_6, LOW);
  digitalWrite(led_7, HIGH);
  digitalWrite(led_8, HIGH);

  }
  if(valeur_recue == 3)            // "si la valeur reçue est égale... enfin... tu connais la
                                     //suite...
  {

  digitalWrite(led_1, HIGH);
  digitalWrite(led_2, LOW);
  digitalWrite(led_3, LOW);
  digitalWrite(led_4, HIGH);
  digitalWrite(led_5, LOW);
  digitalWrite(led_6, LOW);
  digitalWrite(led_7, LOW);
  digitalWrite(led_8, HIGH);

  }
  if(valeur_recue == 4)
  {

  digitalWrite(led_1, HIGH);
  digitalWrite(led_2, HIGH);
  digitalWrite(led_3, LOW);
  digitalWrite(led_4, LOW);
  digitalWrite(led_5, HIGH);
  digitalWrite(led_6, LOW);
  digitalWrite(led_7, LOW);
  digitalWrite(led_8, HIGH);

  }

   if(valeur_recue == 5)
  {

  digitalWrite(led_1, HIGH);
  digitalWrite(led_2, LOW);
  digitalWrite(led_3, LOW);
  digitalWrite(led_4, LOW);
  digitalWrite(led_5, LOW);
  digitalWrite(led_6, HIGH);
  digitalWrite(led_7, LOW);
  digitalWrite(led_8, HIGH);

  }
  if(valeur_recue == 6)
  {

  digitalWrite(led_1, LOW);
  digitalWrite(led_2, LOW);
  digitalWrite(led_3, LOW);
```

```
   digitalWrite(led_4, LOW);
   digitalWrite(led_5, LOW);
   digitalWrite(led_6, HIGH);
   digitalWrite(led_7, LOW);
   digitalWrite(led_8, HIGH);

   }

   if(valeur_recue == 7)
   {

   digitalWrite(led_1, HIGH);
   digitalWrite(led_2, HIGH);
   digitalWrite(led_3, HIGH);
   digitalWrite(led_4, HIGH);
   digitalWrite(led_5, LOW);
   digitalWrite(led_6, LOW);
   digitalWrite(led_7, LOW);
   digitalWrite(led_8, HIGH);

   }

   if(valeur_recue == 8)
   {

   digitalWrite(led_1, LOW);
   digitalWrite(led_2, LOW);
   digitalWrite(led_3, LOW);
   digitalWrite(led_4, LOW);
   digitalWrite(led_5, LOW);
   digitalWrite(led_6, LOW);
   digitalWrite(led_7, LOW);
   digitalWrite(led_8, LOW);

   }

      if(valeur_recue == 9)
   {

   digitalWrite(led_1, HIGH);
   digitalWrite(led_2, LOW);
   digitalWrite(led_3, LOW);
   digitalWrite(led_4, LOW);
   digitalWrite(led_5, LOW);
   digitalWrite(led_6, LOW);
   digitalWrite(led_7, LOW);
   digitalWrite(led_8, LOW);

   }

      if(valeur_recue == 10)
   {

   digitalWrite(led_1, LOW);
   digitalWrite(led_2, LOW);
   digitalWrite(led_3, HIGH);
   digitalWrite(led_4, LOW);
   digitalWrite(led_5, LOW);
   digitalWrite(led_6, LOW);
   digitalWrite(led_7, LOW);
   digitalWrite(led_8, LOW);

   }
}
```

Le code se termine avec le chiffre 0, à défaut du 10.
Le code n'est pas optimisé; on pourrait se passer de répéter les digitalWrite qui sont en position LOW plusieurs fois de suite. Néanmoins, cela permet de pouvoir comprendre et analyser le code bien plus simplement; à commencer par les élèves.

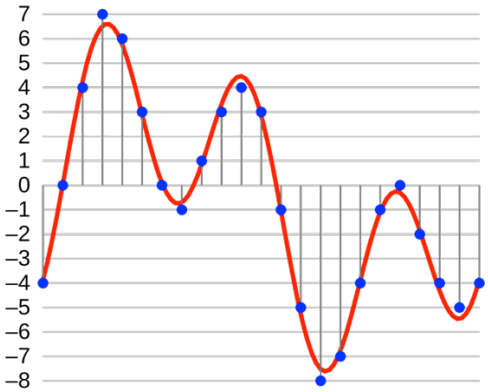
Projet 9: les inputs analogiques

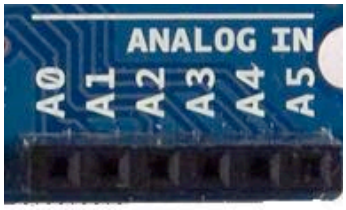
Un signal analogique[13] varie de façon continue au cours du temps. Sa valeur est donc un nombre réel. On trouve des signaux analogiques constamment, comme la température, la vitesse…

Signal numérique

Signal analogique

L'Arduino Uno possède 6 entrées analogiques, numérotées A0 à A5. En réalité, le microcontrôleur n'est pas capable de comprendre un signal analogique. Il faut donc d'abord le convertir en signal numérique par un circuit spécial appelé convertisseur analogique/numérique. Ce convertisseur va échantillonner le signal reçu sous la forme d'une variation de tension et le transformer en valeurs comprises entre 0 et 1023. Attention à ne pas faire entrer une tension supérieure à 5V, ce qui détruirait l'Arduino.

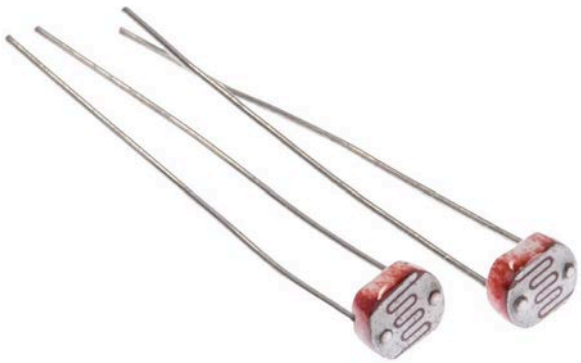




La photorésistance

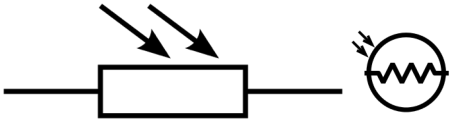
Une photorésistance est un composant électronique dont la résistance (en Ohm Ω) varie en fonction de l'intensité lumineuse. Plus la luminosité est élevée, plus la résistance est basse. On peut donc l'utiliser comme capteur lumineux pour:

- Mesure de la lumière ambiante pour une station météo.
- Détecteur de lumière dans une pièce.
- Suiveur de lumière dans un robot.
- Détecteur de passage.
- …



[13] Plus d'informations: http://f.hypotheses.org/wp-content/blogs.dir/904/files/2013/03/infographie_chloe_manceau.pdf

Ses symboles électroniques sont les suivants:



L'avantage des photorésistances est qu'elles sont très bon marché. Par contre, leur réaction à la lumière est différente pour chaque photorésistance, même quand elles ont été produites dans le même lot. On peut ainsi noter une différence de résistance de plus de 50% entre deux photorésistances du même modèle pour la même luminosité. On ne peut donc pas l'utiliser pour une mesure précise de la lumière. Par contre, elles sont idéales pour mesurer des changements simples de la luminosité.

Circuit 11: diviseur de tension

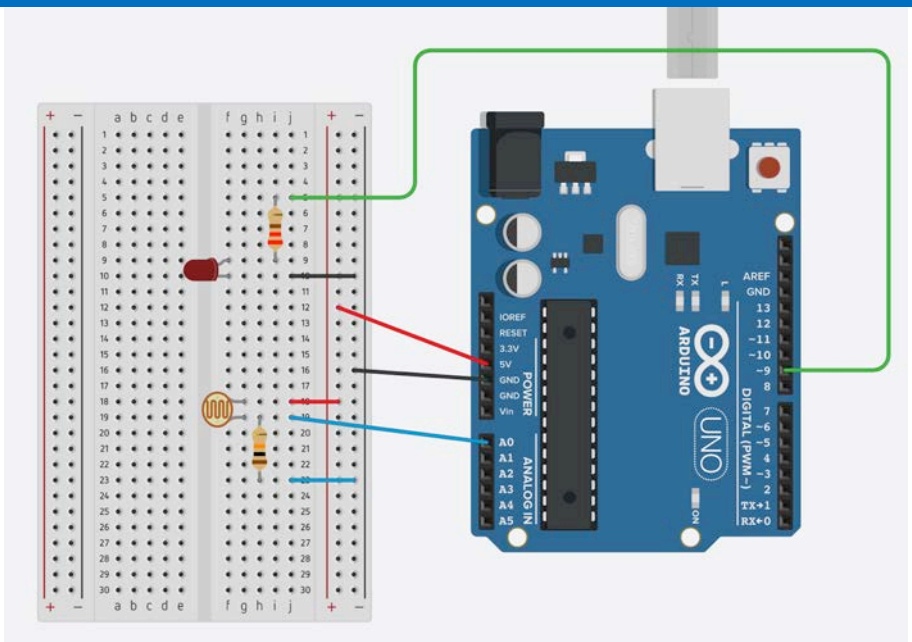
Pour le circuit 11, nous aurons besoin d'une photorésistance que nous connecterons à la broche A0.

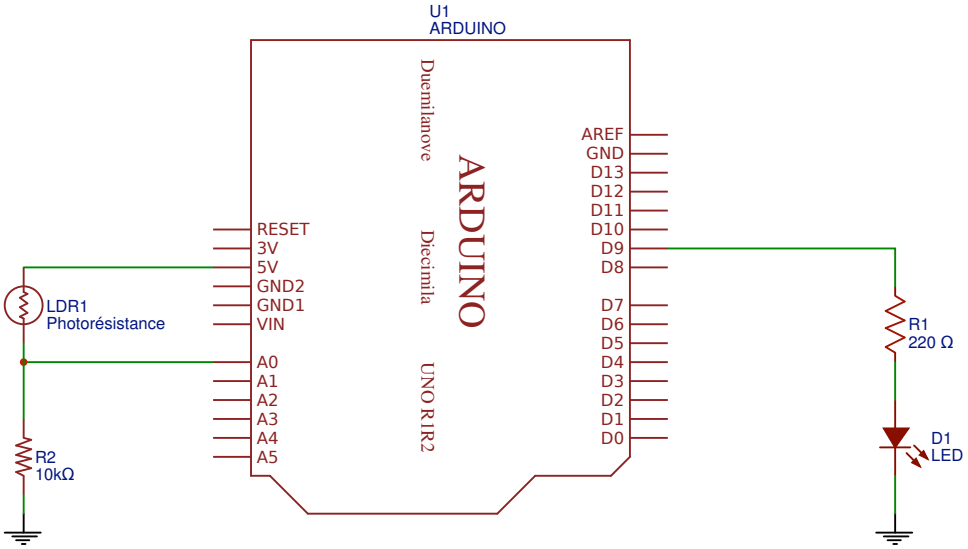
Le montage résistance fixe – photorésistance constitue ce qu'on appelle un **diviseur de tension** . 5V sont introduits dans le circuit depuis l'Arduino.On relie le point entre les deux résistances à une broche analogique de l'Arduino et on mesure la tension par la fonction analogRead (broche). Tout changement de la tension mesurée est dû à la photorésistance puisque c'est la seule qui change dans ce circuit, en fonction de la lumière.

Liste des composants

- 1 LED
- 1 résistance de 220Ω
- 1 résistance de 10kΩ
- 1 photorésistance

Circuit 11





Code 16: valeur de seuil

L'objectif de ce code est de définir un seuil de luminosité au-delà duquel on éteint une LED. Nous allons pour cela utiliser une nouvelle instruction: *if ... else* (*si ... sinon*). C'est donc une *condition*.

Voici ce qui va se passer:

Si la luminosité dépasse le seuil (if (analogRead (lightPin)> seuil)), éteindre la LED (digitalWrite (ledPin, LOW));sinon (else), allumer la LED (digitalWrite (ledPin, HIGH)).

L'instruction == vérifie si deux expressions sont égales. Si elles le sont, alors le résultat sera vrai (*true*) sinon le résultat sera faux (*false*).

Dans ce cas, le seuil est fixé à 900 (sur 1023). En variant la valeur du seuil, on modifie la sensibilité de détection de la luminosité.

```
/*
  Code 16 - Edurobot.ch, destiné à l'Arduino
  Objectif: Eteindre une LED dès que la luminosité est suffisante
*/


//***** EN-TETE DECLARATIVE *****


   int lightPin = 0;   //On renomme la broche A0 en "lightPin"
   int ledPin = 9;     //On renomme la broche 9 en "ledPin"


//***** FONCTION SETUP = Code d'initialisation *****
// La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme


void setup()
  {
   pinMode (ledPin, OUTPUT);
   Serial.begin(9600);
  }
```

```
void loop()
{
int seuil = 900;                          //On définit un seuil de luminosité (sur 1023) à
                                              //partir duquel la LED s'éteint
    if (analogRead (lightPin)> seuil)     //Si la luminosité est plus élevée que le seuil...
     {
     digitalWrite (ledPin, LOW);          //... alors on éteint la LED.
     }
    else                                  //Sinon...
    {
     digitalWrite (ledPin, HIGH);         //...on allume la LED
    }
}
```

Code 17: variation de la luminosité d'une LED en fonction de la lumière ambiante

Imaginons maintenant que l'on veut faire varier la luminosité d'une LED en fonction de la luminosité de la pièce dans laquelle elle se trouve: plus il faut jour, moins la LED est lumineuse.

Théoriquement, cela n'a rien de compliqué; il suffit de combiner le signal de la photorésistance avec la valeur PWM de la LED. Pratiquement, on se retrouve avec un signal en entrée (la photorésistance) découpé en 1024 paliers, alors que ceux du signal PWM comptent 256 paliers. Seulement, 1024, c'est 256 x 4. En divisant la valeur de la variable lightReading par 4, on la rend compatible avec les 256 paliers de variation de luminosité de la LED.

```
/*
  Code 17 - Edurobot.ch, destiné à l'Arduino
  Objectif: Faire varier la luminosité de la LED en fonction de la luminosité de la pièce
  Source: http://learn.adafruit.com/lights
*/


int lightPin = 0;

int lightReading;

int ledPin = 9;
int ledBrightness;


void setup() {

  Serial.begin(9600);
}


void loop() {

lightReading = analogRead(lightPin);


  Serial.print("Lecture analogique = ");
  Serial.println(lightReading);


  lightReading = 1023 - lightReading;

  ledBrightness = lightReading / 4;
  analogWrite(ledPin, ledBrightness);

  delay(100);
}
```

Analyse du code

Premièrement, on va définir nos constantes et nos variables. Ainsi, la broche A0, sur laquelle la photorésistance (couplée à la résistance pulldown) est connectée est renommée lightPin. On créé ensuite une variable nommée lightReading pour y stocker la valeur analogique (comprise entre 0 et 1023) de la valeur de la photorésistance avec le diviseur de tension. Naturellement, la broche 9, sur laquelle est connectée la LED est appelée ledPin. A noter que la broche 9 est compatible *PWM.* Enfin, on créé une variable ledBrightness pour y stocker la valeur de luminosité de la LED (comprise entre 0 et 255).

```
int lightPin = 0;
int lightReading;
int ledPin = 9;
int ledBrightnes
```

Ensuite, on stocke la valeur de la photorésistance avec un analogRead dans la variable lightReading.

```
  lightReading = analogRead(lightPin);
```

Petit aparté: on peut contrôler le bon fonctionnement de la photorésistance en envoyant à la console d'Arduino IDE les valeurs mesurées grâce à Serial.print, qui se trouve dans ***Menu Outils -> Moniteur série***. Ce code est donc facultatif.

```
  Serial.print("Lecture analogique = ");
  Serial.println(lightReading);
```

```
  // Plus il fait sombre, plus la LED doit être brillante.
  // Cela signifique qu'il faut inverser la valeur lue de 0-1023 à 1023-0, avec une
soustraction.
  lightReading = 1023 - lightReading;

  //Enfin il faut faire correspondre les valeurs lues de 0-1023 à 0-255 qui sont les valeurs
utilisées par analogWrite.
  //Pour cela, on convertit la valeur de la variable "lightReading" en la divisant par 4.
  ledBrightness = lightReading / 4;
  analogWrite(ledPin, ledBrightness); // On allume enfin la LED avec la luminosité stockée
dans la variable "ledBrightness".

  delay(100);
}
```

La variation de luminosité est souvent inversée. Ainsi, une radioréveil est très lumineux lorsqu'il fait jour et peu lumineux durant la nuit. Pour simuler cela, il suffit de supprimer la ligne suivante:

```
lightReading = 1023 - lightReading;
```

Code 18: mappage de données

Le mappage des données est l'association des données appartenant à un avec les données appartenant à un autre ensemble, de manière que l'on puisse passer harmonieusement des premières aux secondes[14].
Dans notre cas, nous avons des données inscrites dans une plage située entre 0 et 1023 que nous devons convertir dans une plage entre 0 et 255, qui correspond à 8 bits.

Si on reprend l'exercice précédent, une petite modification permet de faire le mappage des données.

```
/*
  Code 18 - Edurobot.ch, destiné à l'Arduino
  Objectif: Faire varier la luminosité de la LED en fonction de la luminosité de la pièce avec
mappage de données
```

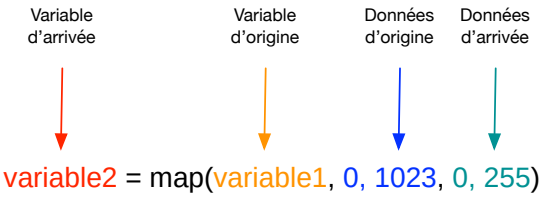
[14] https://fr.wiktionary.org/wiki/mappage et https://www.arduino.cc/reference/en/language/functions/math/map/

```
  Source: http://learn.adafruit.com/lights
*/

int lightPin = 0;     // La photorésistance et la résistance pulldown de 1 à 10K sont
                        connectés à A0
int lightReading;     // Lecture analogique de la valeur de la photorésistance avec le
                                diviseur de tension
int ledPin = 9;       // LED en PWM sur la broche 9
int ledBrightness;     // Variable pour stocker la valeur de luminosité de la LED
void setup() {

  Serial.begin(9600);
}

void loop() {
  lightReading = analogRead(lightPin);

 //Affichage de la valeur de "lightReading" dans la console
  Serial.print("Lecture analogique = ");
  Serial.println(lightReading);     // Affichage de la valeur brute issue de la
                                      //photorésistance.

  // Plus il fait sombre, plus la LED doit être brillante.
  // Cela signifique qu'il faut inverser la valeur lue de 0-1023 à 1023-0, avec une
 //soustraction.
  lightReading = 1023 - lightReading;

  //Enfin il faut faire correspondre avec la fonction "map" les valeurs lues de 0-1023 à 0-255
 //qui sont les valeurs utilisées par analogWrite.

  ledBrightness = map(lightReading, 0, 1023, 0, 255);
  analogWrite(ledPin, ledBrightness); // On allume enfin la LED avec la luminosité stockée
                                      //dans la variable "ledBrightness".

  delay(100);
}
```

On voit à la ligne suivante le mappage:

```
ledBrightness = map(lightReading, 0, 1023, 0, 255);
```

la fonction map fonctionne sous la forme suivante:



Variable1 est l'origine des données. Variable2 est la destination des données.

Projet 10: le potentiomètre

Le potentiomètre est une résistance variable. Contrairement, c'est le bouton de réglage du volume sur une radio. La plupart des potentiomètres sont soit rotatifs, soit linéaires.

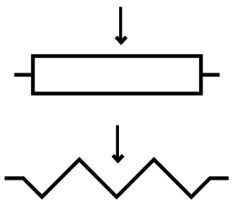
Potentiomètre rotatif

Potentiomètre linéaire

Les potentiomètres sont très fréquents dans les appareils électroniques, par exemple les tables de mixages.

Table de mixage du RadioBus[15]

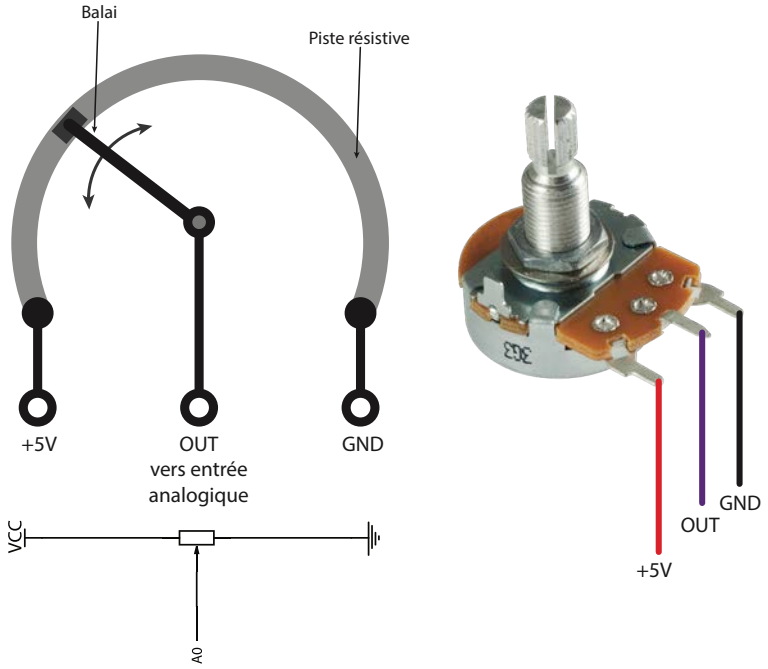
Voici les symboles électroniques (européen dessus et américain dessous) du potentiomètre:



Comme toute résistance, le potentiomètre modifie la tension d'un circuit. On va donc l'utiliser principalement comme entrée (*input*) dans une broche analogique (A0 à A5) de l'Arduino.

[15] http://radiobus.fm

Les potentiomètres ont en général trois broches[16]. Les broches extérieures se connectent sur l'alimentation +5V et sur la terre, alors que la broche centrale envoie le signal sur la broche d'entrée analogique de l'Arduino.

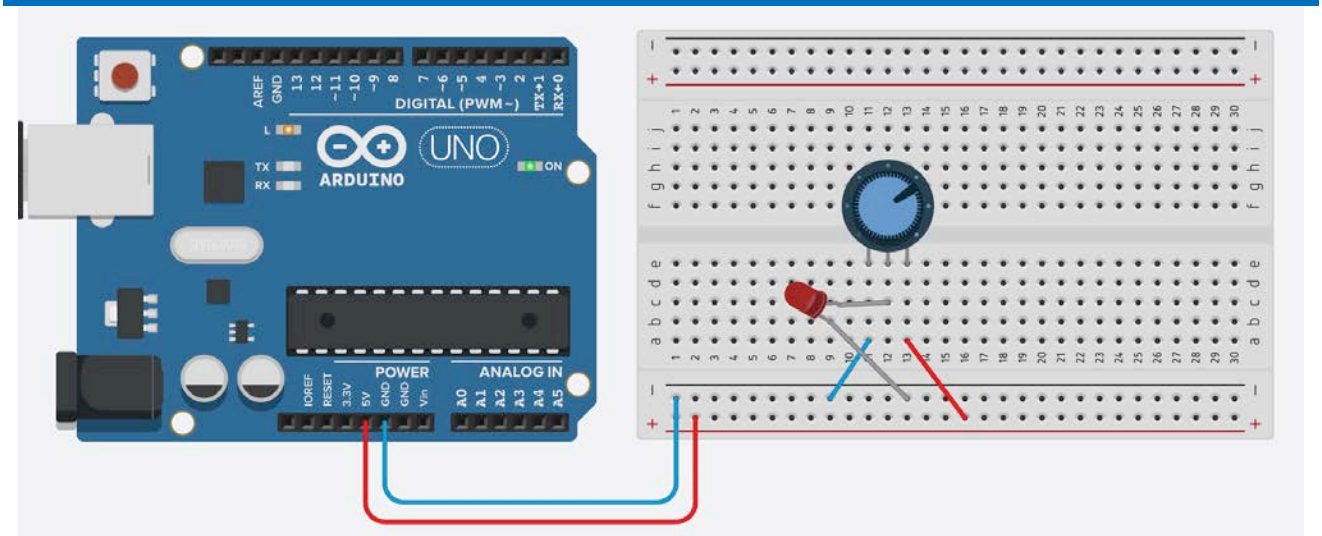


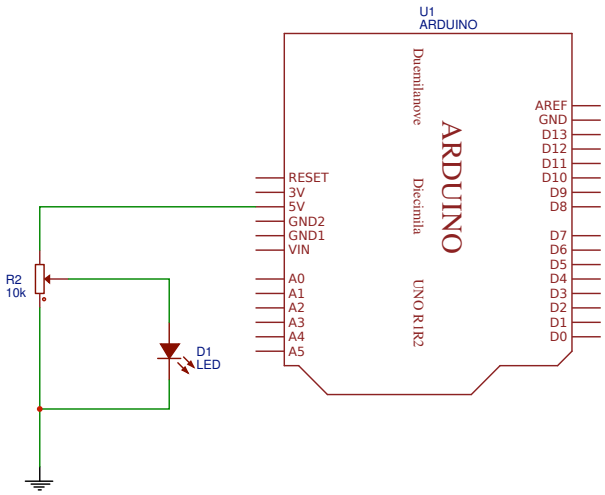
Circuit 12: Utiliser le potentiomètre pour faire varier la luminosité d'une LED

Il est temps de griller notre première LED (si cela n'est pas déjà fait)! C'est le passage obligé de tout bon *maker* ! Pour ce faire, nous allons connecter une LED sur la broche *OUT* du potentiomètre et la mettre à la terre. Petit conseil: avant de brancher, tourner le potentiomètre au minimum, dans le sens inverse des aiguilles d'une montre.

Une fois l'Arduino branché, tourner délicatement le potentiomètre dans le sens des aiguilles d'une montre. La LED va commencer à s'illuminer, puis de plus en plus, avant de… griller.

Circuit 12





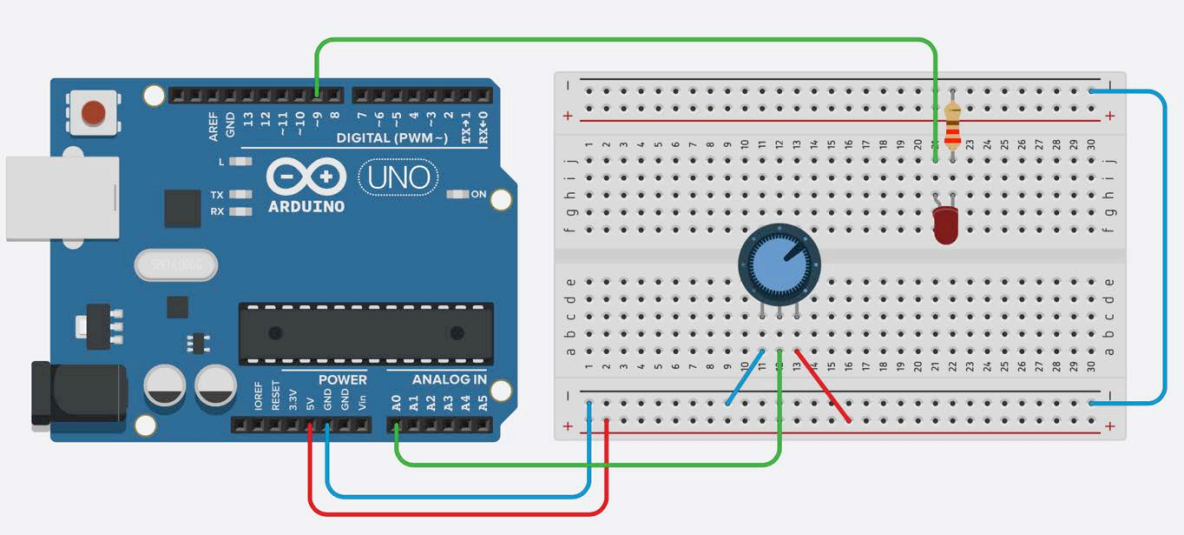
Nous venons de le voir: varier le voltage à l'entrée d'une LED n'est pas la meilleure solution pour faire varier sa luminosité… même si la LED ne grille pas, au-delà de son voltage de fonctionnement, sa durée de vie sera fortement réduite. Mais qu'importe, puisque nous avons la fonction PWM[17]. Nous allons donc coupler le potentiomètre, en entrée analogique sur A0, avec une LED en PWM sur la broche 9.

Liste des composants

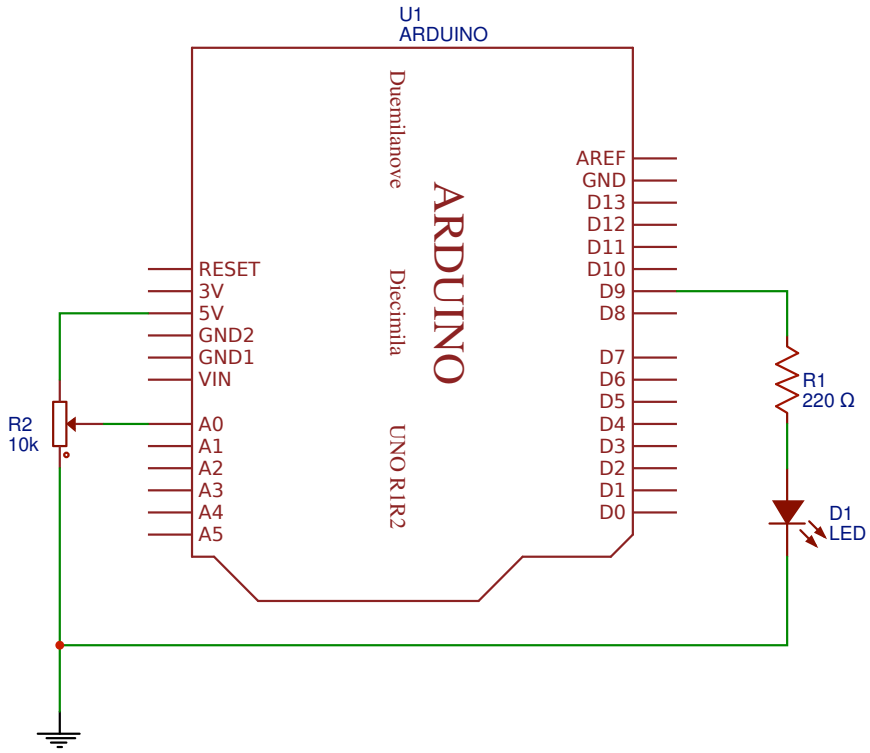
- 1 LED
- 1 résistance de 220Ω
- 1 potentiomètre de 10kΩ ou 20kΩ

Circuit 13: Utiliser le potentiomètre pour faire varier la luminosité d'une LED

Circuit 13



[17] Voir *Projet 7* de ce cours.



Le code est très simple. Comme pour le code 17, on récupère la valeur comprise en 0 et 1023 sur A0, avant de la diviser par 4, afin de la rendre compatible avec la portée du PWM, comprise entre 0 et 255. Comme pour le code 18, on pourrait aussi imaginer un mappage au lieu de la division.

Code 19: Varier la luminosité de la LED en PWM à l'aide d'un potentiomètre

```
/*
  Code 19 - Edurobot.ch, destiné à l'Arduino
  Objectif: Lire la valeur du potentiomètre pour faire varier la luminosité de la LED en PWM
  Source: Philippe Krähenbühl
*/


int ledPin = 9;      //On renomme la broche 9 en "ledPin"
int analogPin = 0;   // Le potentiomètre est connecté à la broche analogue 0
int val = 0;         // la variable qui stocke la valeur lue est mise à 0

void setup()

{
  pinMode(ledPin, OUTPUT);
}

void loop()

{
  val = analogRead(analogPin);   //lecture de la valeur analogue (potentiomètre)
  analogWrite(ledPin, val / 4);  // valeur analogue lue (0 à 1023) divisée par 4 (0 à 255)
}
```

Circuit 14: Allumer les LEDs d'un bargraphe à l'aide d'un potentiomètre

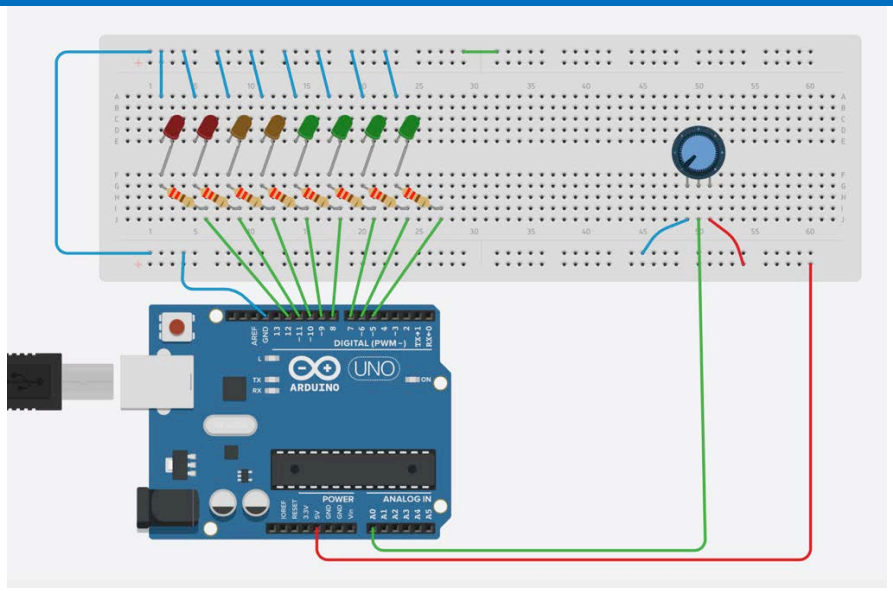
Quand on tourne le bouton pour augmenter le volume, il serait intéressant d'avoir un bargraphe qui nous indique le niveau.

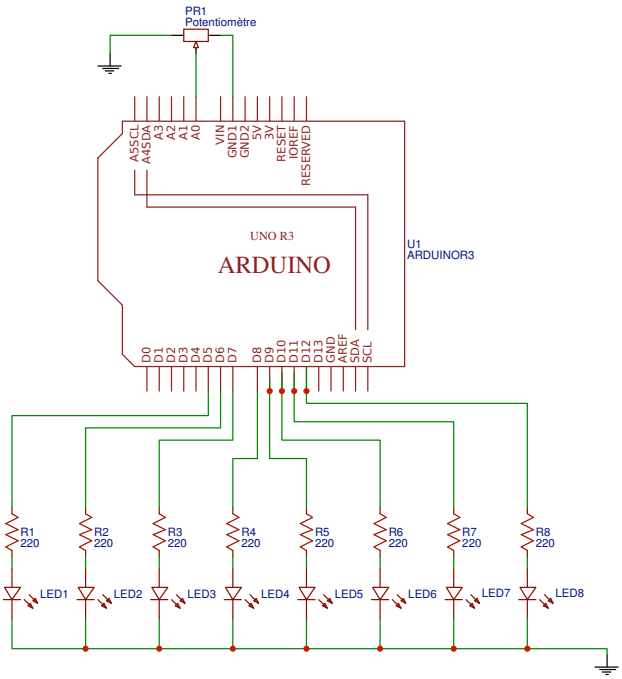
On va donc réaliser un bargraphe à l'aide de 8 LEDs.

Liste des composants

- 8 LED
- 8 résistance de 220Ω
- 1 potentiomètre de 10kΩ ou 20kΩ

Circuit 14





Code 20: Afficher la valeur d'un potentiomètre à l'aide d'un bargraphe

Le cas de ce code est très intéressant. Nous savons maintenant que nous avons 1024 paliers sur l'entrée analogique de l'Arduino. Lorsque l'Arduino reçoit la valeur de 1024, 8 LEDs devront être allumées. Ainsi, lorsqu'il recevra la valeur de 512, 4 LEDs devront être allumées (512 étant naturellement la moitié de 1024). On va donc diviser nos 1024 paliers par 8, pour obtenir l'intervalle qui allumera les LEDs les unes après les autres. Attention; on commence à compter à 0, pas à 1. C'est la raison pour laquelle nos 1024 paliers se terminent à 1023.

On arrive au résultat suivant:

LED1: de 0 à 127	LED2: de 128 à 255	LED3: de 256 à 383	LED4: de 384 à 511
LED5: de 512 à 639	LED6: de 640 à 767	LED7: de 768 à 895	LED6: de 769 à 1023

A partir de là, il suffit d'aller chercher la valeur reçue, qu'on va stocker dans la variable *readValue*, puis de la faire passer dans 8 tests à partir de *if... else* consécutifs.

Le *serial.print* a pour objectif d'afficher dans le moniteur la valeur reçue par l'Arduino[18].

```
/*
  Code 20 - Edurobot.ch, destiné à l'Arduino
  Objectif: Afficher la valeur d'un potentiomètre à l'aide d'un bargraphe
*/


// Information sur les LEDs

int led1 = 5;
int led2 = 6;
int led3 = 7;
int led4 = 8;
int led5 = 9;
int led6 = 10;
int led7 = 11;
int led8 = 12;



// Information sur le potentiomètre
int potPin = A0;
int readValue;
int writeValue;

void setup() {

  Serial.begin(9600);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
  pinMode(led5, OUTPUT);
  pinMode(led6, OUTPUT);
  pinMode(led7, OUTPUT);
  pinMode(led8, OUTPUT);
}

void loop() {

  /*Ecriture de la valeur du potentiomètre
     dans la variable "readValue"
  */
```

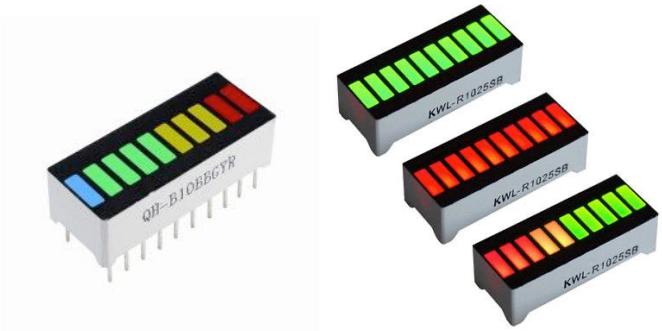
[18] Voir code 17

```
  readValue = analogRead(potPin);

  /*Incription dans le moniteur série
     de "Color=valeur de la variable readValue
  */

  Serial.print("\Val=");
  Serial.print(readValue);
  delay(200);


  // De 0 à 127, led1
  if (readValue < 128) {
    Serial.println("\tled 1");
    digitalWrite(led1, HIGH);
    digitalWrite(led2, HIGH);
    digitalWrite(led3, HIGH);
    digitalWrite(led4, HIGH);
    digitalWrite(led5, HIGH);
    digitalWrite(led6, HIGH);
    digitalWrite(led7, HIGH);
    digitalWrite(led8, HIGH);
  }

  // De 128 à 255, led 1 à 2
  else if (readValue >= 128 && readValue < 256) {
    Serial.println("\tled 2");
    digitalWrite(led1, HIGH);
    digitalWrite(led2, HIGH);
    digitalWrite(led3, LOW);
    digitalWrite(led4, LOW);
    digitalWrite(led5, LOW);
    digitalWrite(led6, LOW);
    digitalWrite(led7, LOW);
    digitalWrite(led8, LOW);
  }

  // De 256 à 383, led 1 à 3
  else if (readValue >= 256 && readValue < 384) {
    Serial.println("\tled 3");
    digitalWrite(led1, HIGH);
    digitalWrite(led2, HIGH);
    digitalWrite(led3, HIGH);
    digitalWrite(led4, LOW);
    digitalWrite(led5, LOW);
    digitalWrite(led6, LOW);
    digitalWrite(led7, LOW);
    digitalWrite(led8, LOW);
  }

  // De 384 à 511, led 4
  else if (readValue >= 384 && readValue < 512) {
    Serial.println("\tled 4");
    digitalWrite(led1, HIGH);
    digitalWrite(led2, HIGH);
    digitalWrite(led3, HIGH);
    digitalWrite(led4, HIGH);
    digitalWrite(led5, LOW);
    digitalWrite(led6, LOW);
    digitalWrite(led7, LOW);
    digitalWrite(led8, LOW);
  }

  // De 512 à 639, led 5
  else if (readValue >= 512 && readValue < 640) {
    Serial.println("\tled 5");
    digitalWrite(led1, HIGH);
    digitalWrite(led2, HIGH);
    digitalWrite(led3, HIGH);
    digitalWrite(led4, HIGH);
```

```
    digitalWrite(led5, HIGH);
    digitalWrite(led6, LOW);
    digitalWrite(led7, LOW);
    digitalWrite(led8, LOW);
  }

  // De 640 à 767, led 6
  else if (readValue >= 640 && readValue < 768) {
    Serial.println("\tled 6");
    digitalWrite(led1, HIGH);
    digitalWrite(led2, HIGH);
    digitalWrite(led3, HIGH);
    digitalWrite(led4, HIGH);
    digitalWrite(led5, HIGH);
    digitalWrite(led6, HIGH);
    digitalWrite(led7, LOW);
    digitalWrite(led8, LOW);
  }

  // De 768 à 895, led 7
  else if (readValue >= 768 && readValue < 896) {
    Serial.println("\tled 7");
    digitalWrite(led1, HIGH);
    digitalWrite(led2, HIGH);
    digitalWrite(led3, HIGH);
    digitalWrite(led4, HIGH);
    digitalWrite(led5, HIGH);
    digitalWrite(led6, HIGH);
    digitalWrite(led7, HIGH);
    digitalWrite(led8, LOW);
  }

  // De 896 à 1023, led 8
  else {
    Serial.println("\tled 8");
    digitalWrite(led1, HIGH);
    digitalWrite(led2, HIGH);
    digitalWrite(led3, HIGH);
    digitalWrite(led4, HIGH);
    digitalWrite(led5, HIGH);
    digitalWrite(led6, HIGH);
    digitalWrite(led7, HIGH);
    digitalWrite(led8, HIGH);
  }
}
```

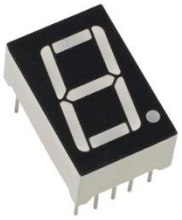
Variante 1: utiliser un bargraphe 10 LEDs

On peut, à ce stade, utiliser un potentiomètre et afficher sa valeur à l'aide d'un bargraphe 10 LEDs. Il suffit pour cela de ne plus diviser 1024 par 8, mais par 10 (en arrondissant).



Variante 2: utiliser un afficheur LED 8 digits

Voici une variante un peu plus sophistiquée. Il s'agit à nouveau d'afficher la valeur d'un potentiomètre. Mais cette fois avec un afficheur LED 8 digits. On affiche la valeur de 0 (min) à 9 (max).



Projet 11 : construire une station météo

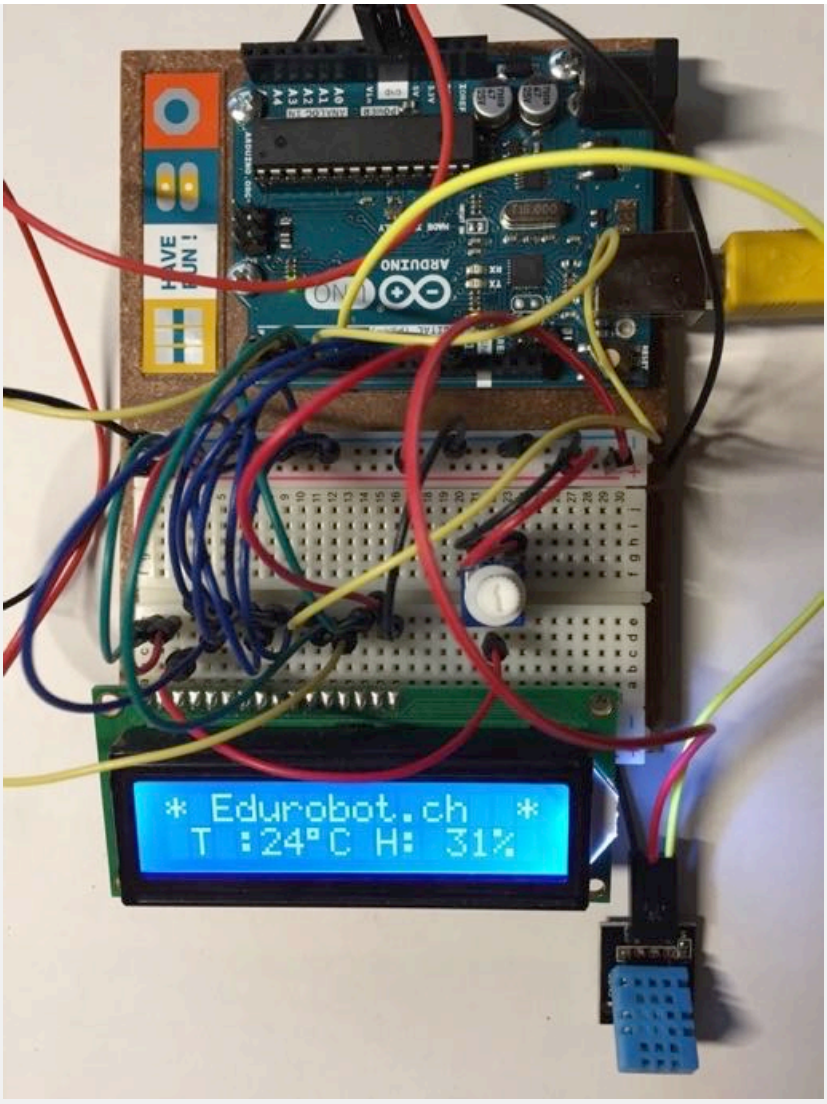
Contribution de **Jean-Pierre Dulex, Ecoles secondaires des Ormonts-Leysin**

Objectif: utiliser un capteur température/humidité et afficher les données sur un écran LCD

Ce projet est la synthèse des input analogiques, avec comme bonus l'utilisation d'un écran pour l'affichage des données.

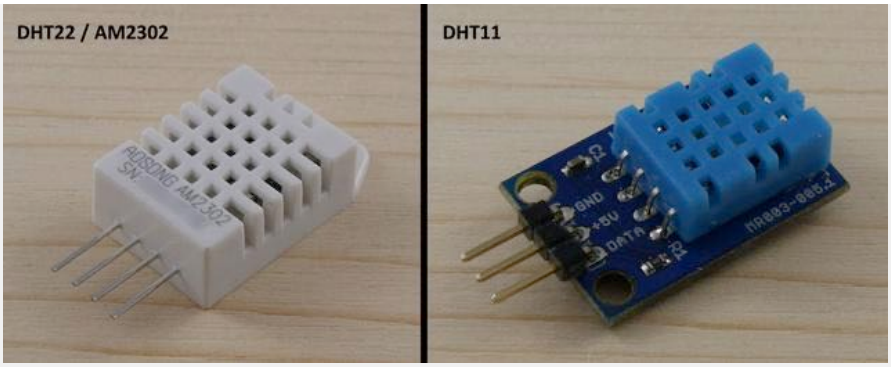
Connaître la température et l'humidité est une information utile pour vivre confortablement, pour stocker des aliments ou du matériel, pour des prévisions météo… Avec ce projet nous apprendrons comment mesurer ces deux données et à les afficher sur un écran LCD.

Avec les projets précédents, nous avons appris à mesurer des valeurs (intensité lumineuse et température) avec des capteurs analogiques grâce aux entrées analogiques de la carte Arduino qui sont munies d'un composant qui transforme les valeurs analogiques mesurées en valeurs numériques. Mais il existe aussi de nombreux capteurs qui transforment eux-mêmes, les valeurs analogiques mesurées en valeurs digitales exploitables par des moyens informatiques. C'est le cas d'un capteur de température et d'humidité très répandu et bon marché, le DHT11 que l'on va utiliser pour ce projet.



En bas à droite, de couleur bleue, le capteur DHT11, au-dessus l'afficheur LCD de deux lignes de 16 caractères qui indique la température et l'humidité actuelle. On remarque aussi un potentiomètre (résistance variable) sur la partie droite du breadboard dont le rôle est de régler la luminosité de l'écran.

Le capteur DHT11 est fourni avec de nombreux kits pour Arduino, autrement on peut l'acheter pour quelques francs. Le DHT11 ne mesure que des températures positives, entre 0°C et 50°C avec une précision de ± 2°C et des taux d'humidité relative de 20 à 80% avec une précision de ± 5%. Pour des besoins plus pointus ou s'il s'agit de mesurer aussi des températures négatives, il faut se munir de son grand frère, le DHT22, deux fois plus onéreux, mais dont la plage de mesure s'étend de -50°C à +125°C, avec une précision de 0.5°C et de 0 à 100% (précision ± 2%) pour la mesure de l'humidité relative.



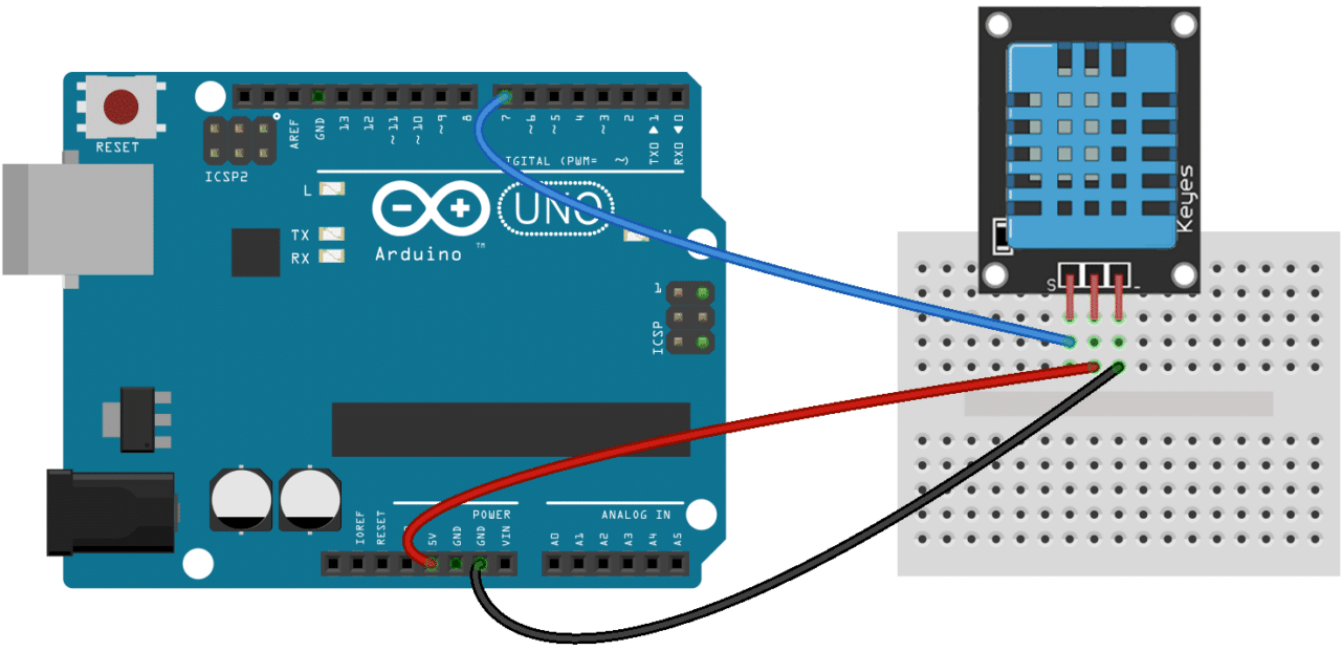
Attention : il existe des versions des capteurs DHTxx qui ne sont pas prémontés sur un petit circuit comme le modèle de gauche ci-dessus utilisé dans ce projet. Si c'est le cas, il faut ajouter une résistance de tirage de 5 à 10 kOhms et un condensateur de 100 nF entre les broches 1 (alimentation 5 V) et 4 (GND). Un schéma du câblage est disponible par ici : https://www.carnetdumaker.net/

Circuit 15

Utilisation du capteur de température et d'humidité

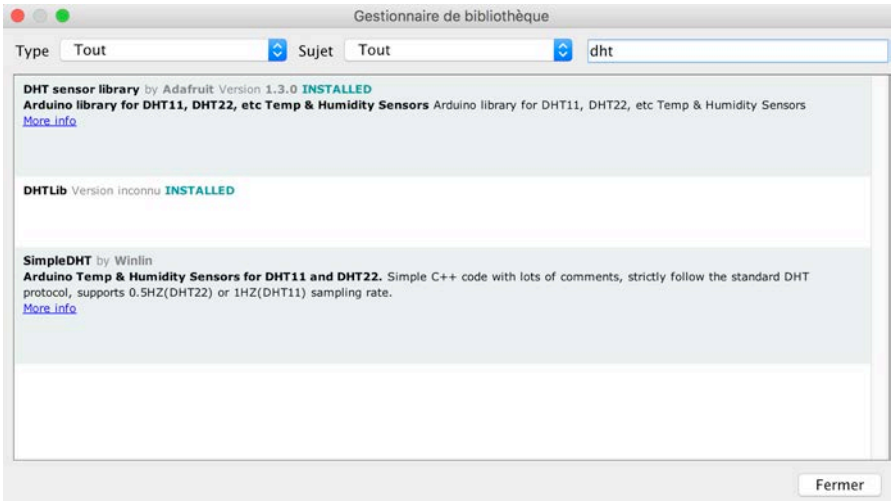
Quand on tient le capteur avec la grille en face de nous, **la broche centrale** est l'alimentation, elle doit être reliée au **5V de l'Arduino**. **La broche la plus à droite** sera reliée au **– (GND).** Quant à **la broche la plus à gauche,** c'est **la broche de données**, elle sera connectée à **l'entrée digitale** utilisée pour lire les données.

Une fois ces données connues, le montage est un jeu d'enfant et un schéma est ici inutile, nous allons choisir le Pin7 pour les données :



Code 21: Acquérir les données du capteur et les afficher.

L'objectif de ce code est d'acquérir via l'une des broches digitales de l'Arduino, les données de température et d'humidité transmises par le capteur DHT11, puis de les afficher sur l'écran de l'ordinateur. Pour communiquer avec le capteur, il faut dans un premier temps télécharger et installer la bibliothèque spécifique à ce capteur (ou plutôt l'une des bibliothèques pour ce capteur, car il y en a plusieurs versions). Les bibliothèques sont des ensembles de fonctions qui s'ajoutent aux fonctions de base du logiciel Arduino.IDE. Certaines bibliothèques sont préinstallées, par exemple celle intitulée LiquidCrystal dont nous aurons besoin plus tard pour l'affichage sur l'écran LCD, d'autres doivent être installées par l'utilisateur au fur et à mesure de ses besoins. La bibliothèque DHTLib que nous allons utiliser pour ce projet fait partie de cette dernière catégorie et doit donc être installée « à la main ».
Depuis la version 1.6.0 de l'IDE Arduino, un gestionnaire de bibliothèque a permis de grandement simplifier l'installation et la mise à jour des bibliothèques. Le gestionnaire est accessible via le menu ***Croquis*** -> ***Inclure une bibliothèque*** -> ***Gérer les bibliothèques***. Une fenêtre de gestion des bibliothèques apparaît dans laquelle il est possible de rechercher une nouvelle bibliothèque en tapant son nom dans le champ de recherche en haut à gauche ou de faire des mises à jour des bibliothèques existantes.



Ci-dessus, en tapant dht dans le champ de recherche on obtient en seconde position la bibliothèque DHTLib que nous allons utiliser. Il ne reste plus qu'à l'installer en cliquant sur le bouton qui apparaît sur la droite quand on la sélectionne.

Une solution alternative pour installer une bibliothèque consiste à télécharger le fichier .zip de la bibliothèque désirée, puis de l'installer via le menu ***Croquis*** -> ***Inclure une bibliothèque*** -> ***Ajouter une bibliothèque.zip.***

La bibliothèque DHTLib peut être téléchargée ici :
http://www.circuitbasics.com/wp-content/uploads/2015/10/DHTLib.zip

***Note** : sur les postes élèves, les bibliothèques supplémentaires sont installées dans le dossier personnel de l'utilisateur et ne seront donc accessibles que pour ce seul utilisateur, il faut disposer d'un accès administrateur pour installer une bibliothèque pour tous les utilisateurs d'un ordinateur.*

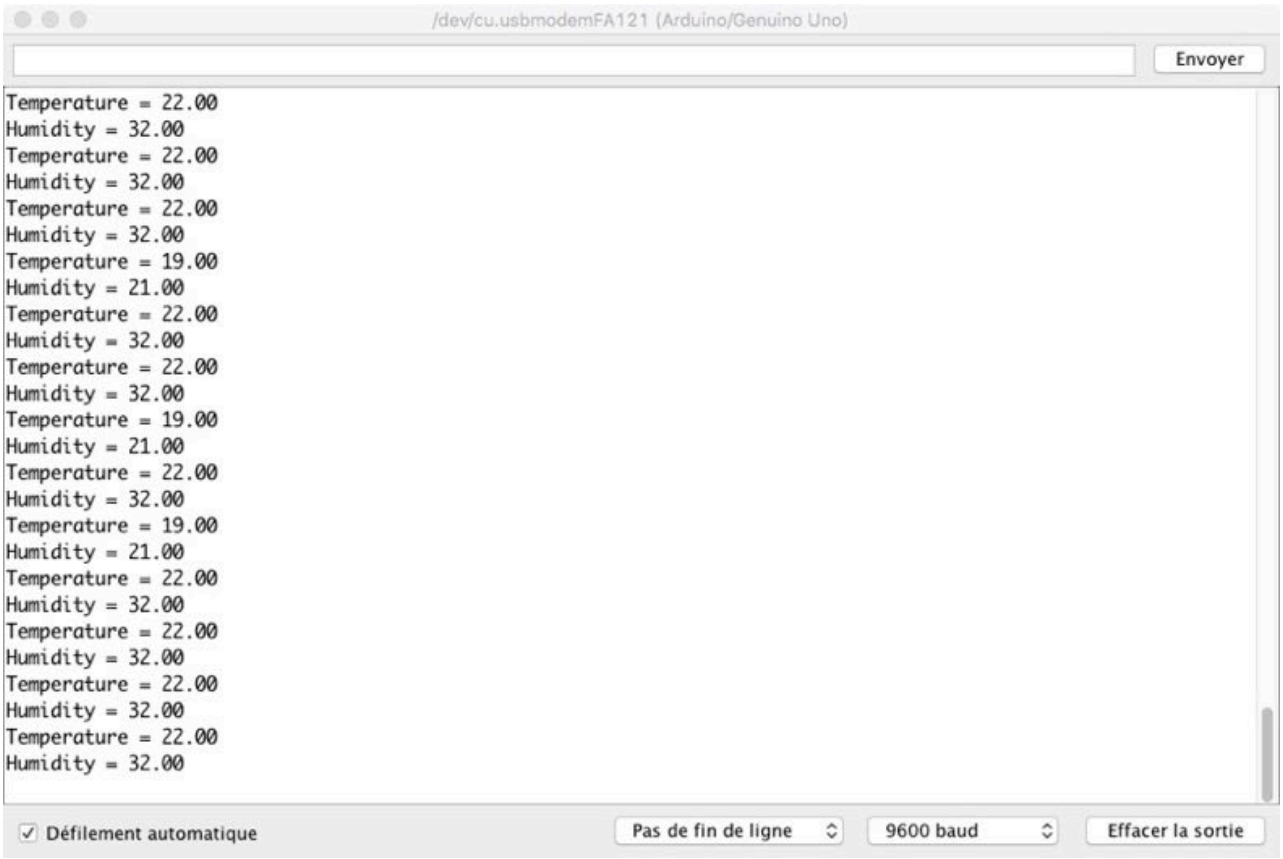
Attention, pour que la bibliothèque nouvellement installée soit utilisable, il faut quitter puis relancer Arduino

La bibliothèque ad hoc étant maintenant installée, on peut passer au code :

```
/*
  Code 21 - Edurobot.ch, destiné à l'Arduino
  Objectif: Afficher la température et l'humidité sur un écran
*/


//***** EN-TETE DECLARATIVE *****
```

```
#include <dht.h>//on charge la bibliothèque

dht DHT;        //on crée l'objet du capteur DHT11

#define DHT11_PIN 7     //on définit le Pin qui sera utilisé pour recevoir les données

void setup()            // début de la fonction setup()
{
  Serial.begin(9600);   //Pour une fois, l'ouverture du port série et la définition de sa
vitesse en bauds est utile, même indispensable puisque l'on va utiliser ce port pour afficher
les valeurs du capteur.
}

//***** FONCTION LOOP = Code d'initialisation *****
// La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme


void loop()       // début de la fonction loop()
{
  int chk = DHT.read11(DHT11_PIN);    // on lit l'état du capteur
  Serial.print("Temperature = ");     //on écrit le mot Température sur le moniteur
  Serial.println(DHT.temperature);    //on affiche sur la même ligne la température lue
  Serial.print("Humidite = "); //on écrit le mot Humidité sur le moniteur
  Serial.println(DHT.humidity);//on affiche sur la même ligne l'humidité lue
  delay(1000);          //on fait une pause de 1 seconde entre 2 interrogations du capteur
}
```

Pour visualiser les données transmises par le capteur, il faut encore ouvrir une fenêtre du moniteur série sur votre ordinateur : ***Menu Outils -> Moniteur série***.

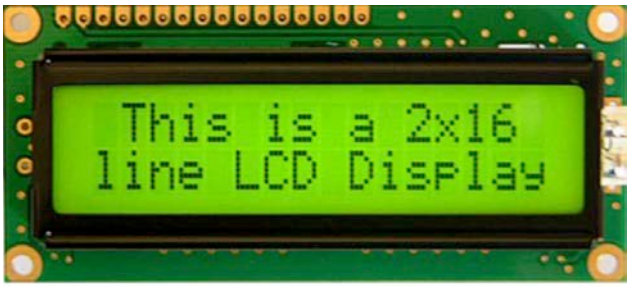


Tant que le moniteur série est ouvert, la température et l'humidité mesurées s'affichent une fois par seconde

Circuit 16

Affichage des valeurs mesurées sur un écran LCD

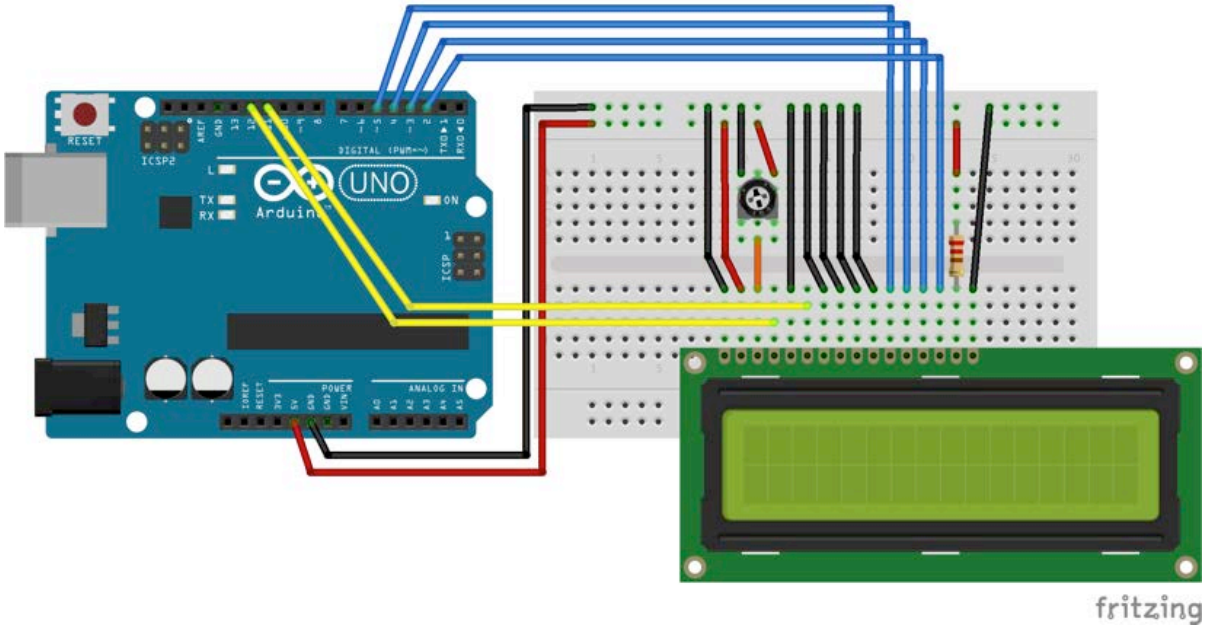
Afficher le résultat des mesures sur le moniteur série est bien sympathique, mais si l'on veut faire une installation de mesure permanente et autonome, cette solution ne convient pas, car il faut que l'ordinateur soit en permanence allumé et branché à la carte. L'affichage sur un petit écran LCD est une solution bien plus élégante et pratique. L'un des écrans LCD les plus répandu dans le monde Arduino et qui est souvent fourni avec les kits de base est le modèle dit 16x2 LCD (il est capable d'afficher deux lignes de 16 caractères chacun). A l'achat un tel écran coûte entre 7 et 15€.



Note : le but final étant d'afficher la température et l'humidité transmise par le capteur DHT11, il est conseillé de laisser le circuit 9 en place (sur un côté ou une extrémité de la breadboard, autrement il faudra le recâbler plus tard. Les images ci-dessous ne comprennent volontairement pas le DHT11, toujours branché à la Pin7 afin d'améliorer leur lisibilité.

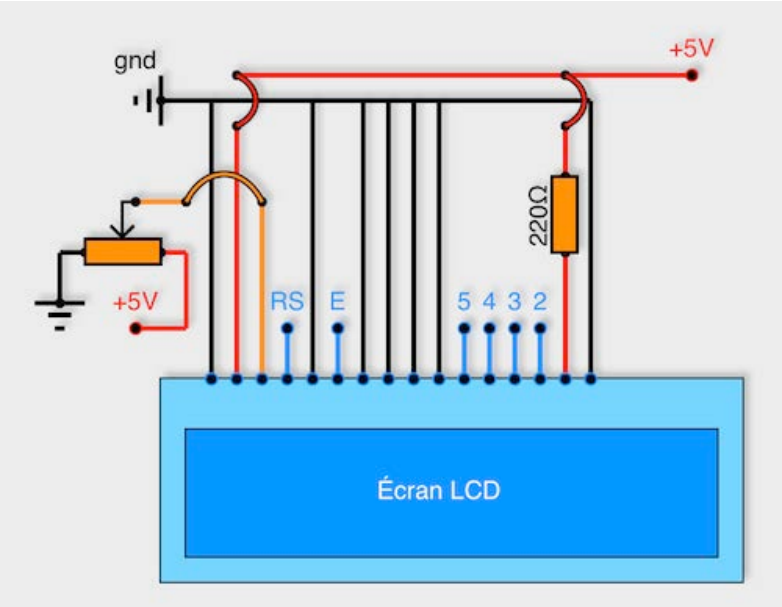
Liste des composants :

- 1 écran LED 2x12
- 1 potentiomètre (résistance variable)
- 1 résistance de 220 Ω
- 1 capteur DHT11(déjà câblé depuis le circuit 9)



Il existe une grande variété de modèles d'écran LCD, certains forts pratiques ayant déjà des pins qui dépassent à la face inférieure et qui s'encastrent directement dans la breadboard, d'autres nécessitant l'usage de connecteurs mâles-mâles pour pouvoir se connecter à la breadboard ou directement à la carte Arduino via des câbles de liaisons.

Tous les modèles d'écran LCD sont dotés de 16 Pins dont le câblage est le suivant :



- Les deux premiers pins tout à gauche servent à l'alimentation de l'écran. Pôle négatif ou GND pour le premier et pôle positif (5V) pour le 2ème.
- Le 3ème pin est connecté à un potentiomètre et sert à régler le contraste de l'écran LCD.
- Le 4ème, noté RS pour Register Select, est connecté au pin 12 de l'Arduino. Il sert à sélectionner la zone mémoire de l'écran LCD dans laquelle nous allons écrire.
- Le 5ème doit être connecté au ground (GND).
- Le 6ème, noté E pour Enable, est connecté au pin 11 de l'Arduino. Il permet de lancer ou non l'écriture dans les zones mémoires.
- Les quatre suivants (7, 8, 9 et 10ème) sont reliés au ground (GND).
- Les quatre qui suivent (11 à 14ème, notés 5, 4, 3, 2 sur le schéma ci-dessus, car ils se connectent sur les Pins 5, 4, 3, 2 de l'Arduino. Ils servent pour la transmission des données à afficher.
- Les deux pins tout à droite (15 et 16ème) servent pour alimenter la LED du rétroéclairage de l'écran LCD. Attention l'avant-dernier (pôle positif 5V) doit impérativement être protégé par une résistance d'environ 220 Ω. Le dernier est relié au pôle négatif (GND).

Une fois le montage effectué et l'Arduino branché au port USB de l'ordinateur, vous pouvez tourner le potentiomètre et vous verrez le contraste de l'écran se modifier.

Code 22: Afficher les données sur l'écran LCD

Contrairement à la bibliothèque «**DHTLib»** utilisée auparavant, la bibliothèque «**LiquidCrystal**»est-elle incluse avec le logiciel Arduino IDE, mais il faut tout de même l'installer avant de pouvoir l'utiliser.

Menu : ***Croquis -> Inclure une bibliothèque -> Gérer les bibliothèques***

Dans l'onglet de recherche, tapez : "LiquidCrystal". Si la bibliothèque est déjà installée, vous verrez noté "INSTALLED" à côté de son nom, autrement cliquez sur **Install**

Attention, pour que la bibliothèque nouvellement installée soit utilisable, il faut quitter puis relancer Arduino IDE.

```
/*
  Code 22 - Edurobot.ch, destiné à l'Arduino
  Objectif: Afficher la température et l'humidité sur un écran LCD
*/

//***** EN-TETE DECLARATIVE *****
```

```
#include <dht.h>        //on inclut la bibliothèque pour le capteur DHT11
#include <LiquidCrystal.h>     //on inclut la bibliothèque pour l’écran LCD

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);//on crée l’objet LCD et on définit les Pins utilisés

dht DHT;                // on crée l’objet du capteur DHT11

#define DHT11_PIN 7             //on définit le Pin utilisé pour les données du DHT11

void setup(){
  lcd.begin(16, 2);             //on initialise la communication avec l’écran LCD
}

void loop()
{
  int chk = DHT.read11(DHT11_PIN);     //on lit les données du capteur DHT
  lcd.setCursor(0,0);   //on place le curseur de l'écran LCD au début de la 1ère ligne
  lcd.print("Temp: ");          //on écrit le mot "Temp: " à l'emplacement du curseur
  lcd.print(DHT.temperature,1);//on écrit la température lue par le capteur, avec 1 chiffre
derrière la virgule
  lcd.print((char)223); //on ajoute le symbole ° après la valeur de la température
  lcd.print("C");               //on ajoute la lettre C pour degré Celsius
  lcd.setCursor(0,1);           //on déplace le curseur de l'écran LCD au début de la 2èmeligne
  lcd.print("Humidity: ");      //on écrit le mot "Hum. rel: " à l'emplacement du curseur
  lcd.print(DHT.humidity,1);   //on écrit l'humidité relative lue par le capteur, avec 1
chiffre derrière la virgule
  lcd.print("%");               //on ajoute le symbole "%" après la valeur de l'humidité
  delay(1000);                  //on attend une seconde avant de procéder à la lecture suivante
}
```

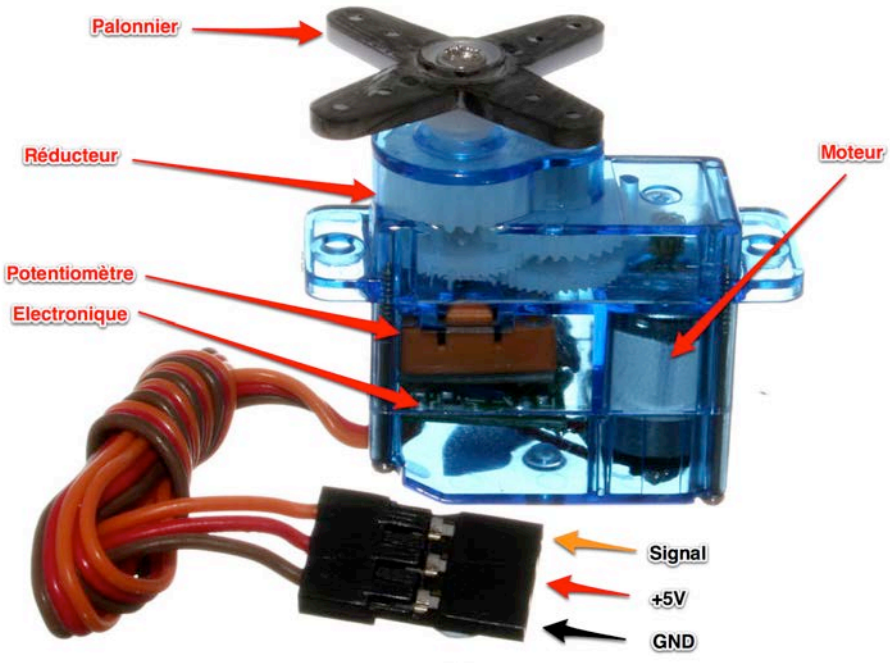
Projet 12 : Utiliser un servomoteur

Les servomoteurs, souvent abrégés en «servo» tout court par leurs utilisateurs, sont des moteurs d'un type particulier, très appréciés pour faire tourner quelque chose jusqu'à une position bien précise et capable de maintenir cette position jusqu'à l'arrivée d'une nouvelle instruction. Ils sont très utilisés dans le modélisme (direction des voitures télécommandées, commande des gouvernes de dérive et de profondeur sur les avions, etc…), mais ont aussi leur place dans la robotique et l'industrie par exemple dans des vannes pour réguler des flux de liquides.

Un servomoteur dit «9 grammes» très répandu dans le monde de l'Arduino.

Dans ce chapitre, nous allons apprendre à utiliser le plus répandu des servomoteurs en modélisme et dans la petite électronique, il s'agit des modèles dits 9g, pour 9 grammes. Extérieurement, il se présente sous la forme d'un petit rectangle, avec deux petits rebords sur les côtés pour le fixer solidement et un axe décentré su lequel on peut fixer des bras interchangeables pour assurer la liaison mécanique avec la pièce qui doit bouger. Même s'il existe de servomoteurs à rotations continues, l'immense majorité des modèles sont capables de bouger leur bras sur 180° seulement.

Vu de l'intérieur, un servomoteur est un peu plus complexe qu'il n'en a l'air :

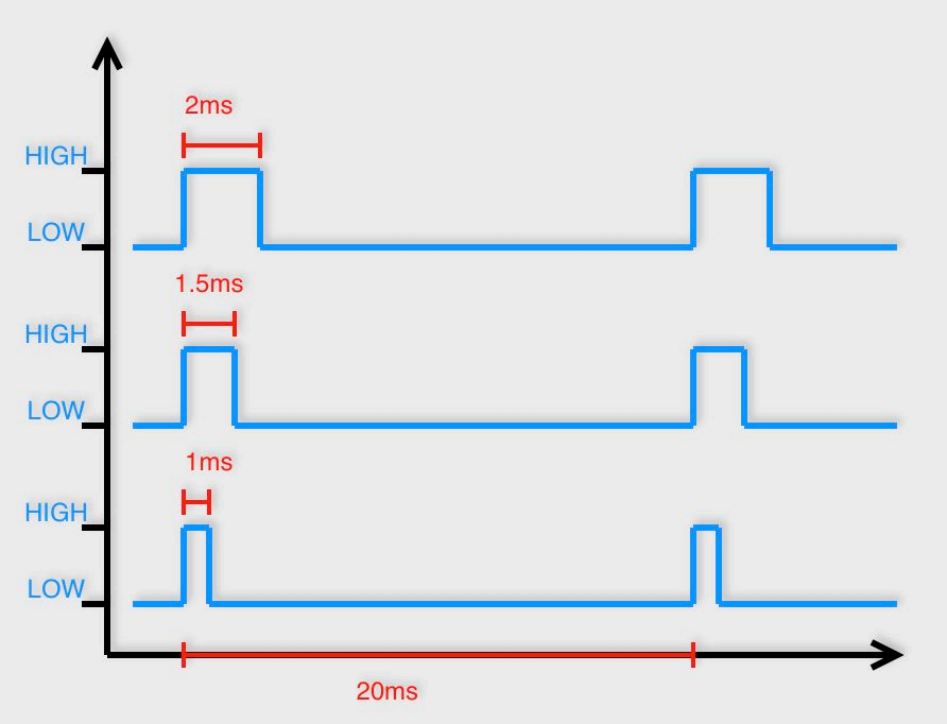


Un petit moteur à courant continu est relié à un potentiomètre (résistance variable) par l'intermédiaire d'un circuit électronique ce qui permet de contrôler finement le moteur en fonction de la position du potentiomètre. Sur l'axe de sortie du moteur, une série d'engrenage permet d'augmenter son couple (sa force utile) en réduisant sa vitesse de rotation.
Quand le moteur tourne, les engrenages s'animent, le bras bouge et entraîne dans son mouvement le potentiomètre. Si le mouvement s'arrête, le circuit électronique ajuste en continu la vitesse du moteur pour que le potentiomètre et donc par extension le bras du moteur reste toujours au même endroit. C'est ce qui permet par exemple à un bras d'un robot de ne pas retomber sous l'effet de son propre poids lorsque le mouvement s'arrête !

Utilisation d'un servomoteur avec l'Arduino

Pour commander un servomoteur, il faut lui envoyer un train d'impulsions dont la période (intervalle de temps entre chaque impulsion) est toujours de 20 ms (millisecondes). Ce qui va varier et qui au final déterminera la position du bras n'est pas la période, mais bien la durée de l'impulsion :

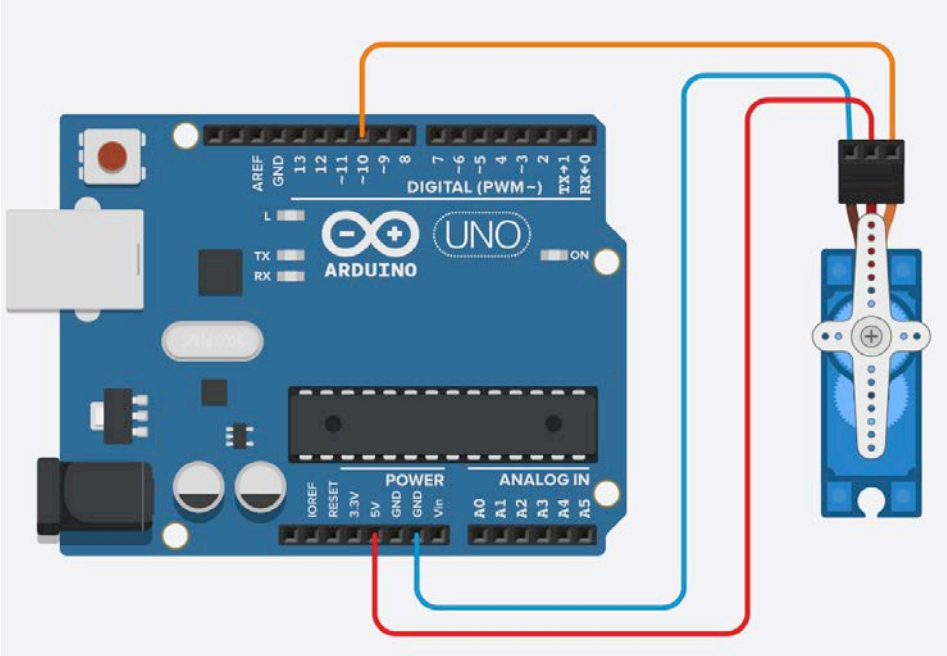
- pour une impulsion de 1 ms, le servomoteur se met en position 0°
- pour une impulsion de 1.5 ms, le servomoteur se met en positon 90°
- pour une impulsion de 2 ms, le servomoteur se met en position 180°



Ce schéma rappelle le PWM que l'on a utilisé pour faire varier l'intensité d'une LED par exemple, le principe en est effectivement très semblable, avec un train d'ondes rectangulaires puisque les données transmises sont digitales (HIGH ou LOW) sans valeurs intermédiaires.

La connexion d'un servomoteur ne pose pas de difficulté. Le fil rouge se connecte à l'alimentation (5V), le fil noir se connecte au ground (GND) et le fil jaune (attention parfois blanc ou orange ou… suivant le matériel dont dispose le fabricant chinois ?) à n'importe quelle sortie numérique de l'Arduino (pin 0 à 13)

Circuit 17



Code 23: Faire bouger le bras d'un servomoteur dans les deux sens

L'objectif des trois codes ci-dessous est de se familiariser avec l'utilisation des servomoteurs.

Pour les 3 codes, nous aurons besoin de la bibliothèque Servo qui fait partie d'office du logiciel Arduino, mais qui n'est pas installée par défaut.

Menu : ***Croquis -> Inclure une bibliothèque -> Servo***

Attention, pour que la bibliothèque nouvellement installée soit utilisable, il faut quitter puis relancer Arduino

```
/*
Code 23 - Edurobot.ch, destiné à l'Arduino
Objectif : Faire bouger le bras d'un servomoteur dans un sens puis dans l'autre, indéfiniment
*/

//*****EN-TETE DECLARATIVE*****

#include <Servo.h>      //on inclut la bibliothèque pour piloter un servomoteur
Servo monServo;   //on créée l'objet monServo

void setup()
{
  monServo.attach(9); //on définit le Pin utilisé par le servomoteur
}

void loop()
{
  for (int position = 0; position <=180; position ++){  //on crée une variable position qui
prend des valeurs entre 0 à 180 degrés
  monServo.write(position);   //le bras du servomoteur prend la position de la variable
position
```

```
  delay(15);      //on attend 15 milliscondes
  }
  for (int position = 180; position >=0; position --){  //cette fois la variable position
passe de 180 à 0°
  monServo.write(position);     //le bras du servomoteur prend la position de la variable
position
  delay(15);    //le bras du servomoteur prend la position de la variable position
  }
}
```

Une fois votre code fonctionnel, n'hésitez pas à tester des délais d'attente différents, de demander des parcours de 90° seulement ou d'autres valeurs, de varier le pas des incréments utilisés, par exemple de 5° en 5°, etc.. Et observez à chaque fois le nouveau résultat.

Nous avons dit en parlant des servomoteurs qu'une fois une position atteinte, le moteur, grâce aux informations, maintient le bras dans la position demandée jusqu'à ce qu'un nouvel ordre lui parvienne. Cette fonction de maintien est primordiale aussi bien en modélisme qu'en robotique. Si un bras robotisé saisit quelque chose par exemple, il ne faut pas qu'il retombe juste sous l'effet du poids de la pièce saisie et de son bras. Pour cela le servomoteur doit donc continuellement continuer d'ajuster la position à maintenir. La petite variation de code ci-dessous nous prouvera d'une part que la position demandée est maintenue même quand on demande à l'Arduino d'effectuer une autre tâche (ici, allumer la diode 13) et vous pouvez aussi essayer de tourner le servo à la main (sans forcer !) pour sentir la résistance à la rotation qu'exerce le servo qui tente de maintenir sa position.

Code 24: Servomoteur et gestion des tâches

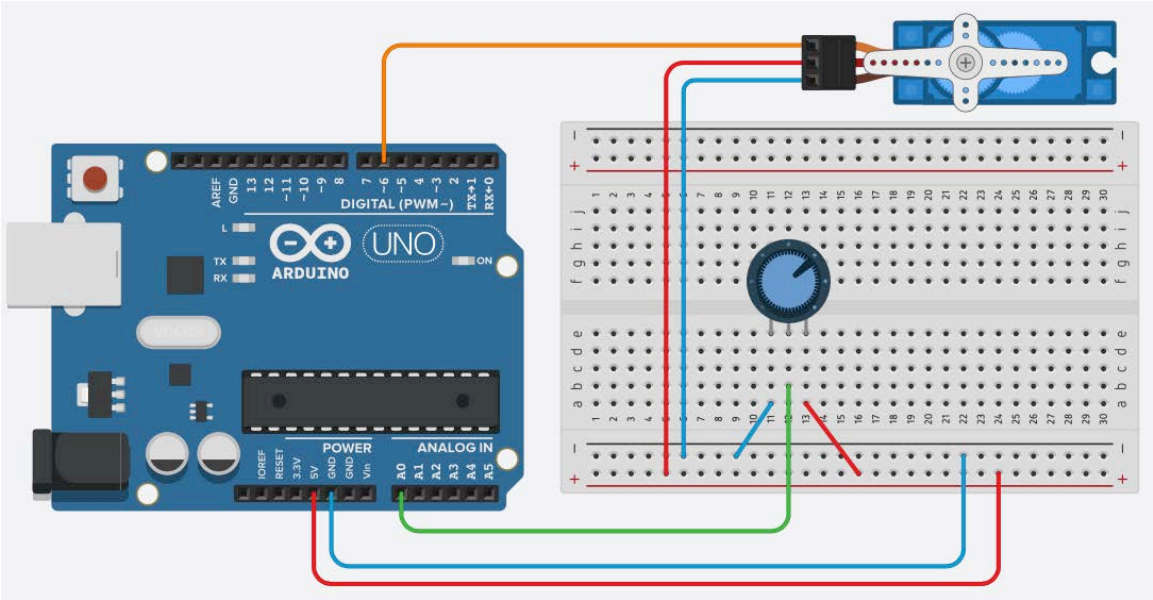
```
/*
Code 24 - Edurobot.ch, destiné à l'Arduino
Objectif : prouver que la bibliothèque Servo permet au servomoteur d'agir et de se maintenir
en position même lorsque l'Arduino effectue une autre tâche.
*/

//*****EN-TETE DECLARATIVE*****

#include <Servo.h>  //on inclut la bibliothèque pour piloter un servomoteur
Servo monServo;   //on créée l'objet monServo

void setup()
{
  monServo.attach(9); //on définit le Pin utilisé par le servomoteur
  pinMode(13,OUTPUT); //la Pin13 est mise en mode OUTPUT
}

void loop()
{
  monServo.write(0); // on dit à l'objet de mettre le servo à 0°
  diode13(); // appel de la fonction diode13 qui est définie plus bas
  monServo.write(180); // on dit à l'objet de mettre le servo à 180°
  diode13(); // appel de la fonction diode13
}
void diode13()  //on va faire clignoter 15 fois la diode 13
{
  for (int t=0;t<15;t++){
    digitalWrite(13,HIGH);
    delay(100);
    digitalWrite(13,LOW);
    delay(100);
} }
```

A vous de faire aussi varier les angles demandés, le nombre de clignotements de la LED 13, le temps d'attente…

Et pour en terminer avec le pilotage des servomoteurs, voici un code qui ne manque pas de provoquer son petit effet. Vous allez ajouter un potentiomètre à votre montage et c'est la position du potentiomètre que vous tournerez qui servira à positionner le bras du servomoteur.

Pour vous aider, voici le schéma du montage :

Circuit 18



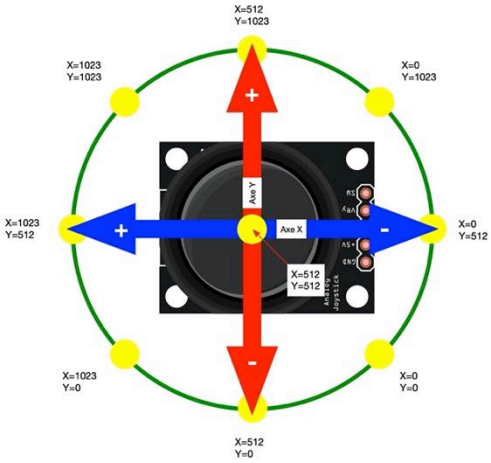
Code 25: commander un servomoteur avec un potentiomètre

Ce code tout simple permet. A l'aide d'un mappage, de lier les 1024 pas d'un potentiomètre aux 180° de rotation d'un servo.

```
/*
Code 25 - Edurobot.ch, destiné à l'Arduino
Objectif : commander un servomoteur avec un potentiomètre
*/

//*****EN-TETE DECLARATIVE*****

#include <Servo.h>  //on inclut la bibliothèque pour piloter un servomoteur
Servo monServo;   //on créée l'objet monServo
int pinmonServo=9; //on définit la Pin9 liée à la commande du servomoteur
int pinPotar=A0;  //on définit la Pin analogique A0 pour la lecture du potentiomètre

void setup()
{
  monServo.attach(pinmonServo);  //on lie l'objet monServo au pin de commande
}

void loop()
{
  int valeurPotar=analogRead(pinPotar); // on lit la valeur du potentiomètre
  int angle=map(valeurPotar, 0,1023,0,180); //on transforme la valeur analogique lue en valeur
                                            //d'angle entre 0 et 180°
  monServo.write(angle); //on met le bras du servomoteur à la position angle
}
```

Code 26: Commander deux servomoteurs à l'aide d'un joystick

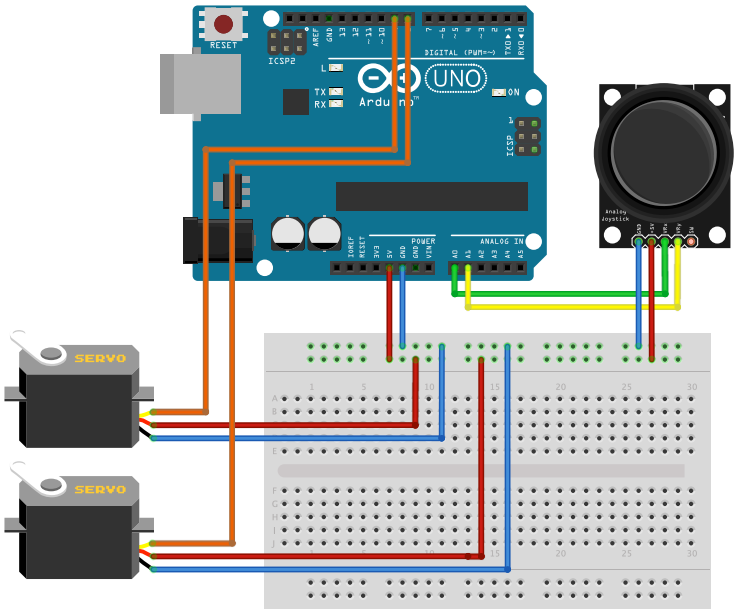
Tout joueur de jeux vidéo connaît le joystick, cette interface de commande. Il se base sur système de coordonnée dans une échelle X et Y. La position sur chaque échelle est déterminée par un potentiomètre. Dans le cas de l'Arduino, on décomposera chaque axe en 1024 pas. Ainsi, le *zéro*, à savoir la position centrale, correspond à 512 sur l'axe X et Y.



En appliquant ce que nous avons appris avec le code 25, il devient possible de commander deux servos avec un joystick.

Pour le câblage, un joystick possède une broche +5V (parfois notée VCC) pour l'alimentation électrique, une broche GND (terre), deux broches VRx et VRy, respectivement pour la position de l'axe X et Y et enfin une broche SW (parfois appelée SEL). Cette dernière correspond au bouton-poussoir, lorsqu'on appuie sur le joystick.

Circuit 19



Le code 26 n'est qu'une adaptation du code 25: il s'agit d'ajouter un second servo.

```
/*
Code 26 - Edurobot.ch, destiné à l'Arduino
Objectif : commander deux servomoteurs avec un joystick
*/

//*****EN-TETE DECLARATIVE*****

#include <Servo.h>  //on inclut la bibliothèque pour piloter un servomoteur
Servo monServo1;   //on créée l'objet monServo1
Servo monServo2;   //on créée l'objet monServo2
int pinmonServo1=8; //on définit la broche 8 liée à la commande du servomoteur
int pinmonServo2=9; //on définit la broche 9 liée à la commande du servomoteur
int AxeX=A0;  //on définit la broche analogique A0 pour la lecture de l'axe X
int AxeY=A1;  //on définit la broche analogique A1 pour la lecture de l'axe Y

void setup()
{
  monServo1.attach(pinmonServo1);  //on lie l'objet monServo1 à la broche pinmonServo1
    monServo2.attach(pinmonServo2);  //on lie l'objet monServo2 à la broche pinmonServo2
}

void loop()
{
  int valeurAxeX=analogRead(AxeX); // on lit la valeur de l'axe X
  int angleX=map(valeurAxeX, 0,1023,0,180); //on transforme la valeur analogique lue en valeur
d'angle entre 0 et 180°
  monServo1.write(angleX); //on met le bras du servomoteur 1 à la position angle
    int valeurAxeY=analogRead(AxeY); // on lit la valeur de ,l'axe Y
  int angleY=map(valeurAxeY, 0,1023,0,180); //on transforme la valeur analogique lue en valeur
d'angle entre 0 et 180°
  monServo2.write(angleY); //on met le bras du servomoteur 2 à la position angle
}
```

Voyons maintenant une autre manière d'obtenir le même résultat. Ici, on stocke les valeurs envoyées par le joystick respectivement dans les variables *X* et *Y*. on utilise ensuite un calcul pour convertir la valeur des deux variables en degrés, à savoir 1024 pas en 180 degrés: 180/1024, ce qui donne une valeur de 0.1756° par pas.

```
/*
Code 27 - Edurobot.ch, destiné à l'Arduino
Objectif : commander deux servomoteurs avec un joystick
Inspiré par SurtrTech.com
*/

##include <Servo.h> //Librarie et déclaration des servos

Servo monservo1;
Servo monservo2;

int AxeY = 1; //Déclaration des pins analogiques utilisés pour brancher l'axe X et Y
int AxeX = 0;

void setup() {
  Serial.begin(9600);
  pinMode(AxeX, INPUT); //Déclaration des modes des pins en INPUT et l'emplacement des servos
  monservo1.attach(8);
  pinMode(AxeY, INPUT);
  monservo2.attach(9);

}
void loop() {
  int X=analogRead(AxeX); //On lit d'abord la valeur analogique depuis le pin du X (0v-
5v)=>(0-1023)
  X=X*0.1756; //on fait une calibration suivant les limites du servo (0-1023)=>(0-180)
  X=180-X;//On utilise ceci pour changer le sens d'orientation du support
  monservo1.write(X); //On écrit la valeur finale calibrée et orientée
```

```
  int Y=analogRead(AxeY);//On lit d'abord la valeur analogique depuis le pin du Y (0v-5v)=>(0-
1023)
  Y=Y*0.1466; //on fait une calibration suivant les limites du servo (0-1023)=>(0-150)
  monservo2.write(Y);//On écrit la valeur finale calibrée
}  Y=Y*0.1466; //on fait une calibration suivant les limites du servo (0-1023)=>(0-150)
  myservo2.write(Y);//On écrit la valeur finale calibrée
}
```

Il devient alors possible de commander un bras robotisé de 4 servos avec simplement deux joysticks. Dans ce cas, il faut prévoir une alimentation externe pour les servos, et ne pas oublier de réaliser une mise à la terre commune entre l'alimentation, l'Arduino, les servos et les joysticks.

Conclusion (provisoire)

Conclusion provisoire, car les possibilités de l'Arduino et de l'électronique sont sans fin. Au fur et à mesure des envies de mes élèves, de nos découvertes et expérimentations, mais aussi des contributions de lecteurs, ce cours est constamment mis à jour.

Ce cours a été rédigé par Frédéric Genevey, enseignant aux Ecoles Primaires et Secondaires d'Ecublens, Suisse, http://mitic.education