edurobot.ch

Arduino à l'école

Cours de l'élève

Édition 2016

VERSION PROVISOIRE 1

Frédéric Genevey
info@edurobot.ch

Ressources Éducatives Libres

Avant-propos

Ce cours peut être téléchargé à l'adresse suivante: http;//edurobot.ch/arduino/intro.pdf

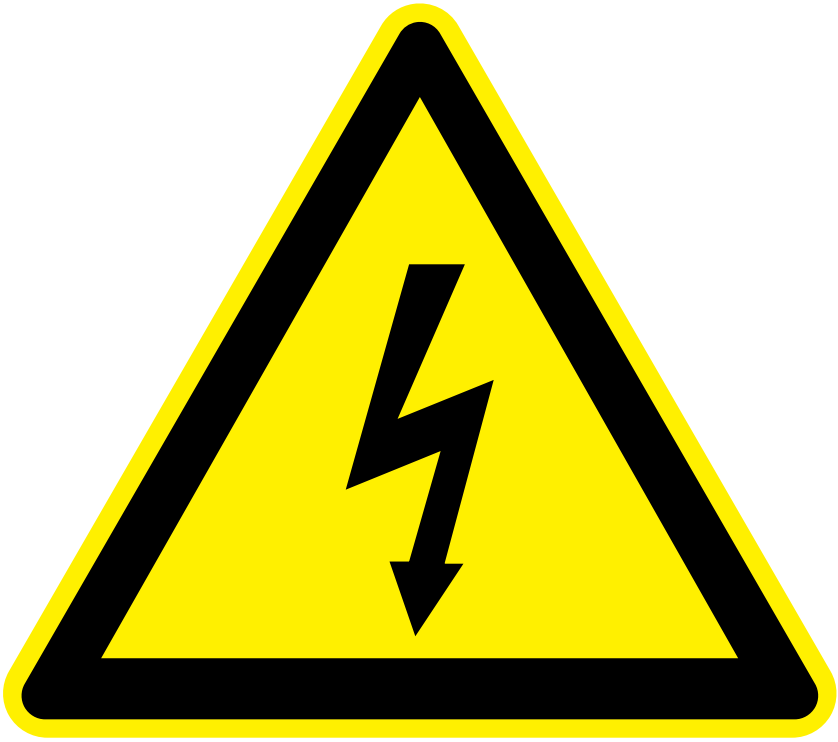
Les codes utilisés dans ce cours peuvent être téléchargés à l'adresse suivante:

http://edurobot.ch/arduino/codes.zip

[1] http://www.wsis-community.org/pg/groups/14358/open-educational-resources-oer
[2] http://creativecommons.org/licenses/by-nc-sa/3.0/ch/

Consignes de sécurité

L'électricité peut être mortelle! Pour éviter tout risque, en particulier avec des élèves, il convient de ne travailler qu'avec de la **très basse tension (TBT)**. La tension de fonctionnement de l'Arduino se situe autour de 5 Volts. Avec les élèves, je me suis volontairement limité à travailler en courant continu avec une différence de potentiel maximum de 24 Volts.



Quelques règles élémentaires de sécurité

- Ne jamais connecter directement l'Arduino sur le secteur (230 Volts alternatifs).
- Pour l'alimentation des projets, utiliser des transformateurs répondants aux normes de sécurité en vigueur.
-
- Ne pas démonter d'appareils électroniques, sans supervision. Certains composants, comme les condensateurs, peuvent délivrer des décharges électriques mortelles, même lorsqu'ils ne sont pas connectés au secteur.

Préface

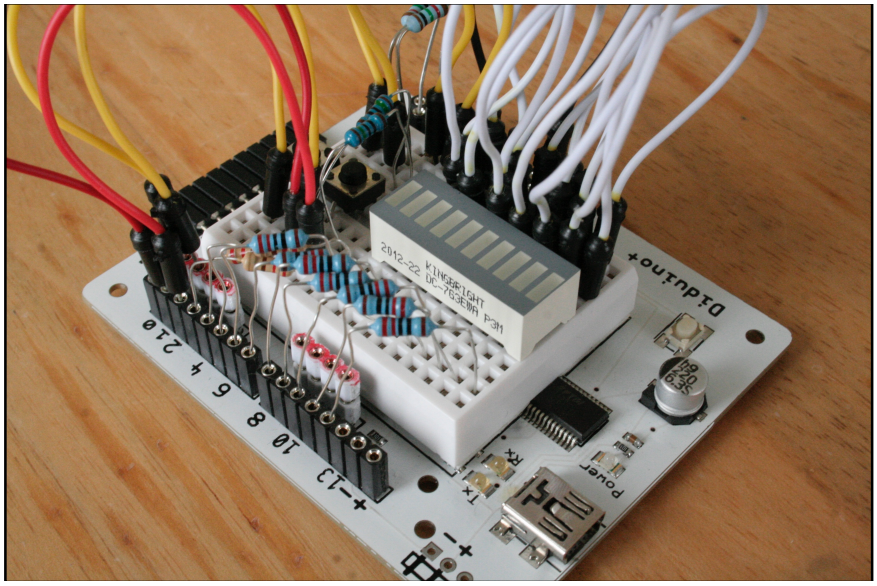
Lorsque Massimo Banzi et ses collègues de l'*Interaction Design Institute* d'Ivrea, en Italie, ont développé l'Arduino, l'objectif était de permettre aux étudiants de pouvoir disposer d'une plateforme valant le prix d'une pizza pour réaliser des projets interactifs[3].
Ainsi, l'Arduino a été conçu dès le départ dans un but pédagogique, pour être bon marché, doté d'une grande quantité d'entrées et de sorties, compatible Mac, Windows et Linux, programmable avec un langage très simple et open source. Il n'y a là que des avantages pour le monde scolaire, en particulier parce que l'Arduino se situe au croisement entre l'informatique, l'électronique et les travaux manuels[4].

L'approche pédagogique de l'Arduino est particulière. Il ne s'agit pas d'aborder la matière d'une manière linéaire, mais en bricolant et en «bidouillant»: on câble, on branche et on regarde ce que cela donne. C'est une approche par la pratique et l'expérimentation, qui convient très bien à des élèves, même (et surtout) peu scolaires. Il y a bien sûr un risque de «griller» un Arduino; mais il ne s'agit que de 30 francs de matériel, et pas d'un ordinateur à 1'200 francs! L'Arduino est un excellent outil pour le *learning by doing* et le *project based learning*. Une approche par la théorie, même si elle reste possible, serait contre-productive.

La meilleure preuve que l'Arduino est parfaitement adapté aux élèves est, qu'en quelques leçons, ils sont déjà prêts à réaliser des projets concrets.

Ce cours a été pensé pour des élèves (et des enseignants) qui n'ont aucune notion en programmation et en électronique. Par rapport au gigantesque potentiel de l'Arduino, il est volontairement limité, mais il s'efforce d'être progressif et surtout axé sur la pratique.

Note: Il n'y a pas de différence marquante entre du matériel Arduino et Genuino.



[3] Histoire de l'Arduino: http://www.framablog.org/index.php/post/2011/12/10/arduino-histoire

[4] Références: http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino et http://www.edurobot.ch/?p=1554

Introduction

Références

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

Sources principales:

- http://arduino.cc/fr/
- http://www.arduino.org
- http://eskimon.fr/
- http://eskimon.fr/ebook-tutoriel-arduino
- http://mediawiki.e-apprendre.net/index.php/Diduino-Robot
- https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino

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://blog.makezine.com/arduino/
- http://www.craslab.org/arduino/livrethtml/LivretArduinoCRAS.html
- http://arduino103.blogspot.ch
- http://www.semageek.com

Ce cours ne permet qu'une introduction à l'électronique. Un cours bien plus complet et très bien fait est disponible ici:

- http://fr.openclassrooms.com/sciences/cours/l-electronique-de-zero

Bibliographie

Il existe de nombreux livres sur Arduino et sur l'électronique. Voici les trois livres que je vous conseille, pour leur côté progressif et didactique:

Electronique:

L'électronique en pratique, de Charles Platt

ISBN: 978-2212135077

L'approche pédagogique de ce livre est l'apprentissage par la pratique. On commence par expérimenter et découvrir, et ensuite seulement vient la théorie pour affermir et expliciter les découvertes. Cela en fait donc un ouvrage de référence pour l'apprentissage de l'électronique à l'école; en particulier en complément des Arduino et Raspberry Pi.

- Feuilleter ce livre sur Amazon.fr
- Commander sur Amazon.fr

Arduino:

Le grand livre d'Arduino, d'Erik Bartmann

ISBN: 978-2212137019

Ce livre est sans doute l'un des meilleurs pour débuter sur Arduino. Il offre une introduction rapide à l'électronique et surtout le code est très bien expliqué, agrémenté de schémas. Il ne se limite enfin pas à quelques notions de base, mais va assez loin.

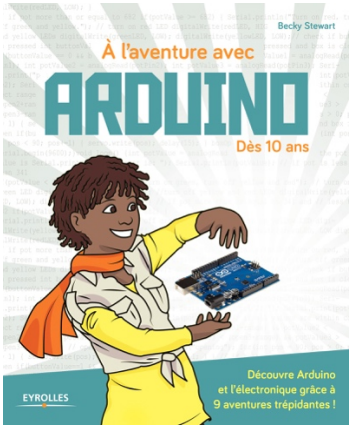
- Feuilleter sur Amazon.fr
- Commander sur Amazon.fr

Démarrez avec Arduino – 2e édition: Principes de base et premiers montages, par Massimo Banzi

ISBN: 978-2100701520

Massimo Banzi est l'un des principaux créateurs d'Arduino. Autant dire qu'il connaît son sujet! L'accent de cet ouvrage est mis sur une initiation à la programmation de l'Arduino, plus que sur l'électronique. A conseiller à tous les débutants.

- Commander le livre sur Amazon.fr



A l'aventure avec Arduino – Dès 10 ans, par Becky Stewart

ISBN: 978-2212143140

Le sous-titre; « *dès 10 ans* » implique que ce livre manque complètement sa cible: gros pavés de textes, présentation dense, touffue et un peu triste ainsi que de devoir attendre la page 41 avant de réaliser son premier montage (une résistance, une LED…
Maintenant, ce livre a quand même des avantages certains, pour un enseignant ou un parent qui va accompagner des enfants: les exercices sont bien agencés, les explications sont claires et simples à comprendre. Les objectifs sont ambitieux, puisqu'on va aller jusqu'à aborder les registres à décalage et les servos.

- Commander le livre sur Amazon.fr

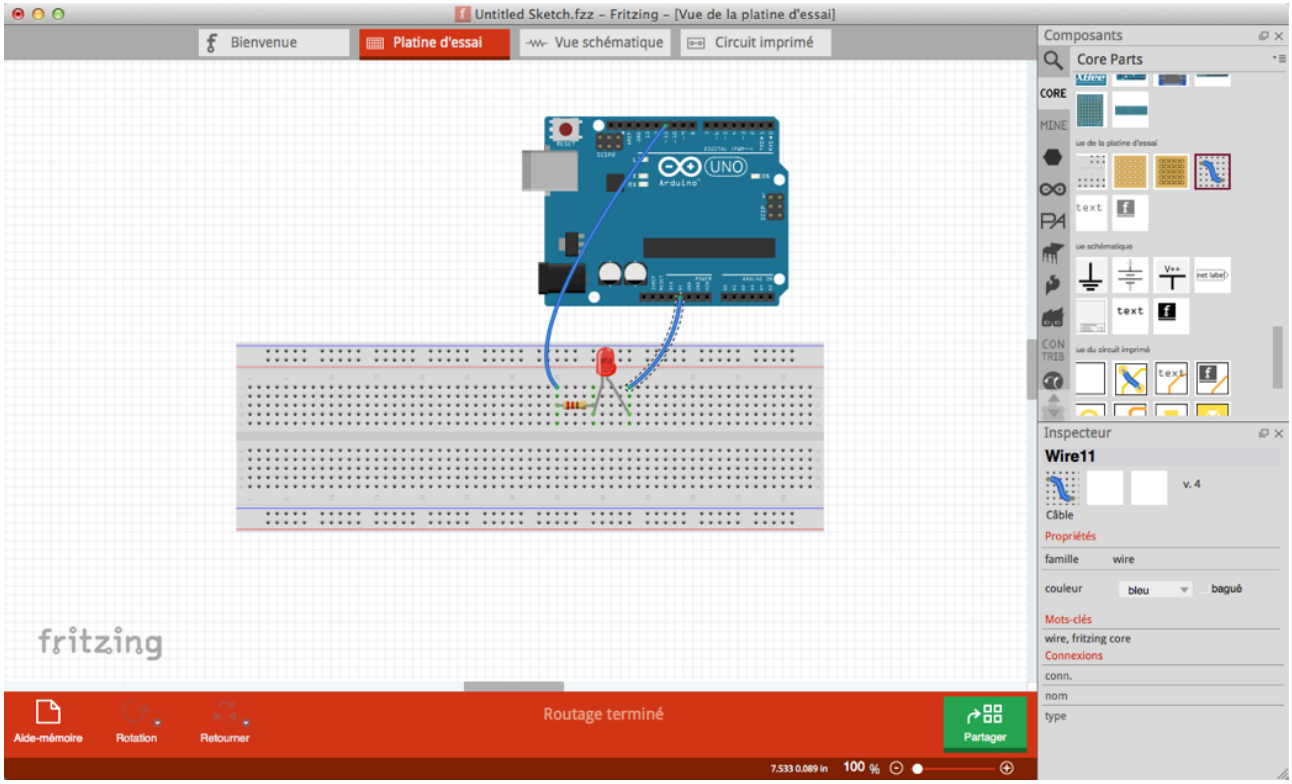
À propos des schémas électroniques

Pour réaliser les schémas électroniques, nous utilisons les outils suivants, disponibles gratuitement sur Mac et PC:

- Solve Elec: http://www.physicsbox.com/indexsolveelec2fr.html
- Fido Cadj: http://davbucci.chez-alice.fr/index.php?argument=elettronica/fidocadj/fidocadj.inc
- Fritzing: http://fritzing.org

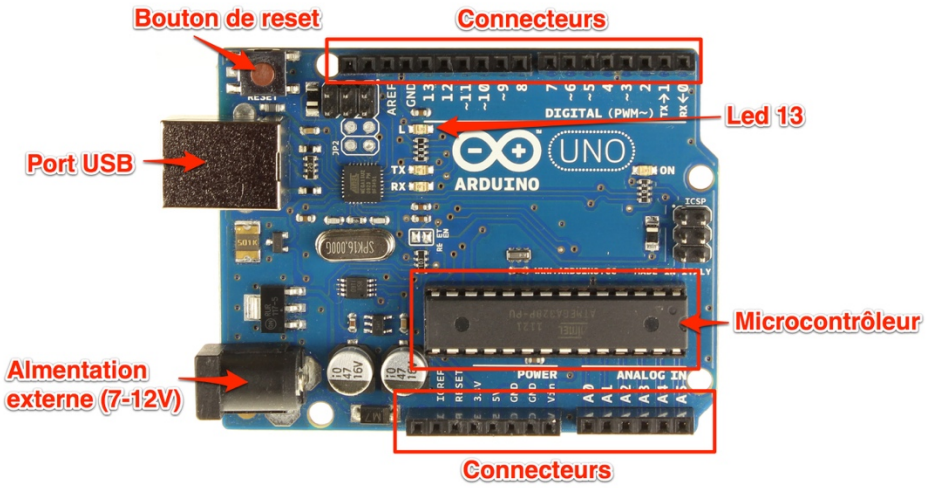
Utilisation de Fritzing

Fritzing est un logiciel gratuit fonctionnant sur Mac, Windows et Linux, développé à l'Université de Potsdam. Il permet à la fois de réaliser des schémas électroniques, mais aussi des vues des montages électroniques. Il inclut par défaut les modules officiels Arduino, ainsi que de nombreux composants des firmes Adafruit et Sparkfun.



Le logiciel Frirtzing peut être téléchargé à l'adresse suivante: http://fritzing.org

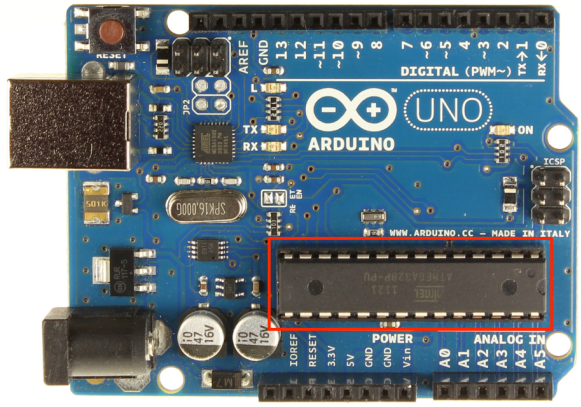
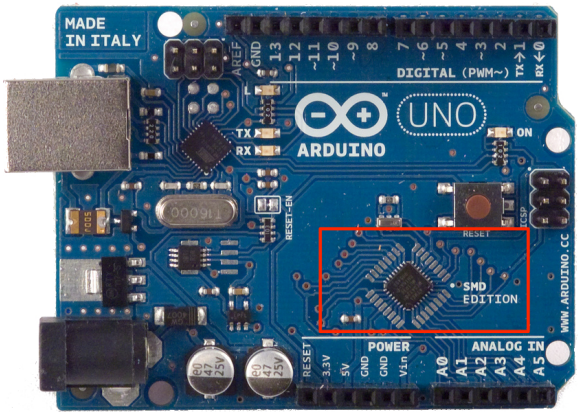
Découverte de la plateforme Arduino



Le microcontrôleur

C'est le cerveau de notre carte. Il va recevoir le programme que nous allons créer et va le stocker dans sa mémoire avant de l'exécuter. Grâce à ce programme, il va savoir faire des choses, qui peuvent être : faire clignoter une LED, afficher des caractères sur un écran, envoyer des données à un ordinateur, mettre en route ou arrêter un moteur…

Il existe deux modèles d'Arduino Uno: l'un avec un microcontrôleur de grande taille, et un autre avec un microcontrôleur dit SMD (SMD: *Surface Mounted Device*, soit composants montés en surface, en opposition aux composants qui traversent la carte électronique et qui sont soudés du côté opposé). D'un point de vue utilisation, il n'y a pas de différence entre les deux types de microcontrôleurs.

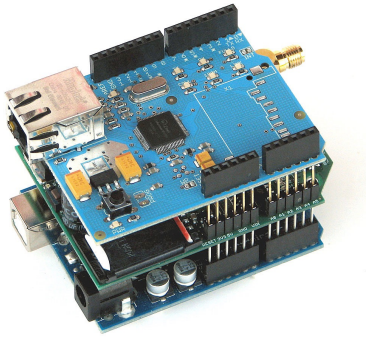
Arduino Uno	Arduino Uno SMD
	

L'alimentation

Pour fonctionner, une carte Arduino a besoin d'une alimentation. Le microcontrôleur fonctionnant sous 5V, la carte peut être alimentée 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** 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

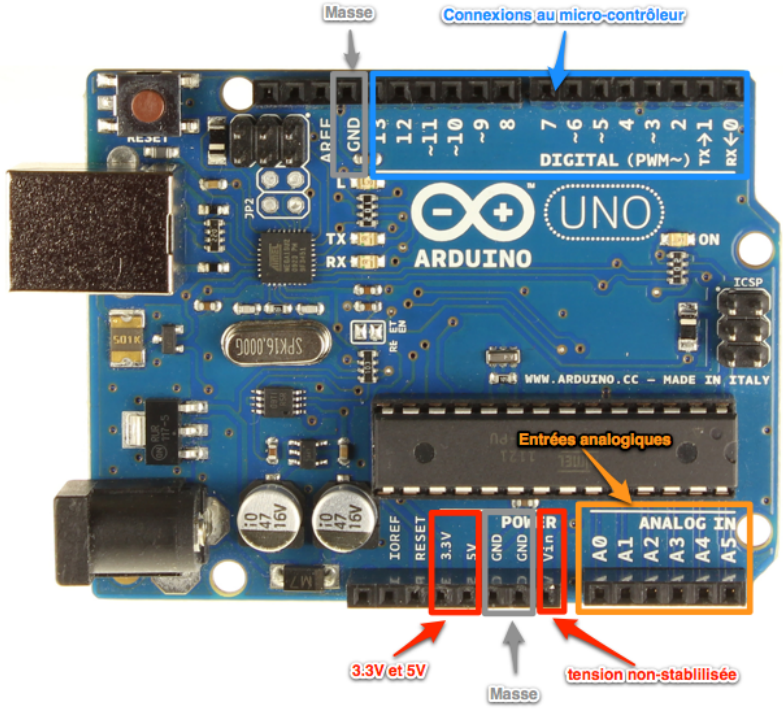


A part une LED sur la broche 13, la carte Arduino ne possède pas de composants (résistances, diodes, moteurs...) qui peuvent être utilisés pour un programme. il est nécessaire de les rajouter. Mais pour cela, il faut les connecter à la carte. C'est là qu'interviennent les connecteurs, aussi appelés **broches**.

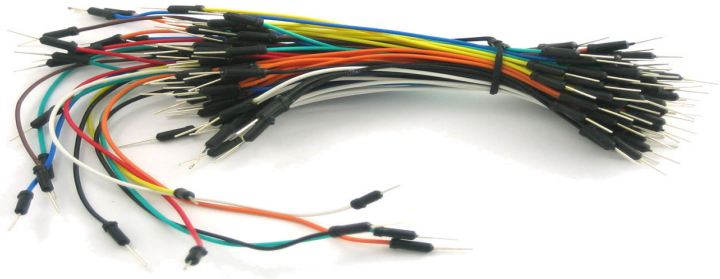
Sur les Arduino et sur beaucoup de cartes compatibles Arduino, les connecteurs se trouvent au même endroit. Cela permet de fixer des cartes d'extension, appelée *shields* en les empilant.

Exploration des connecteurs 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)

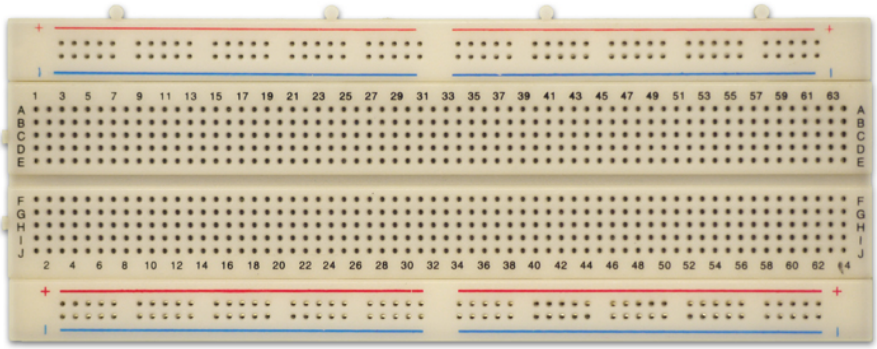


Les connexions entre les composants sont réalisées par des *jumpers*, sortes de petits câbles.

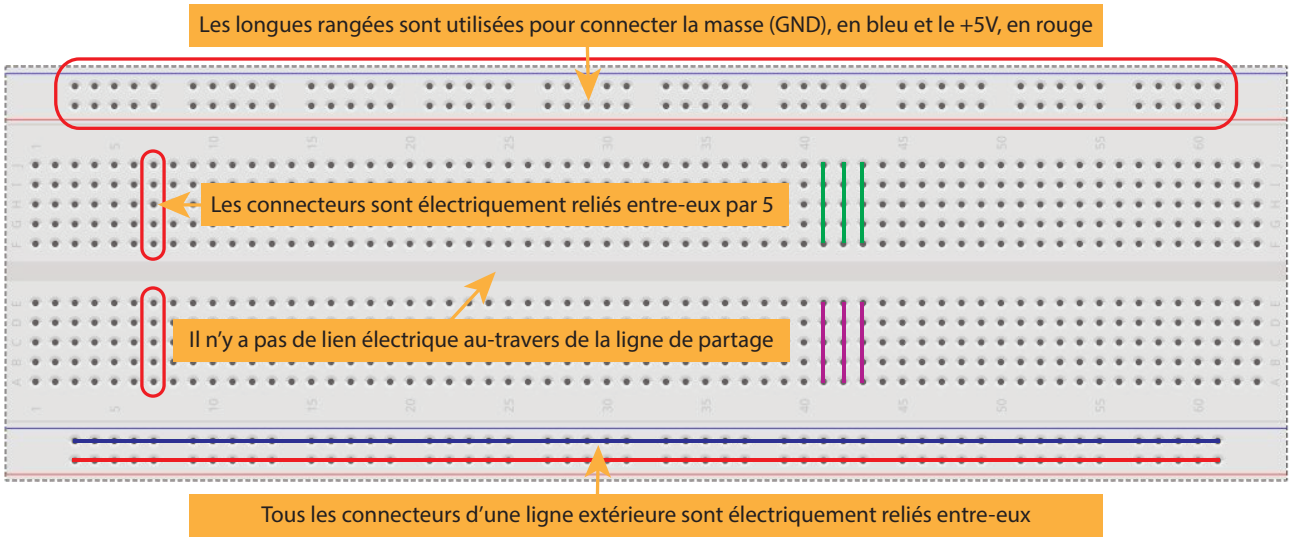


La platine d'expérimentation

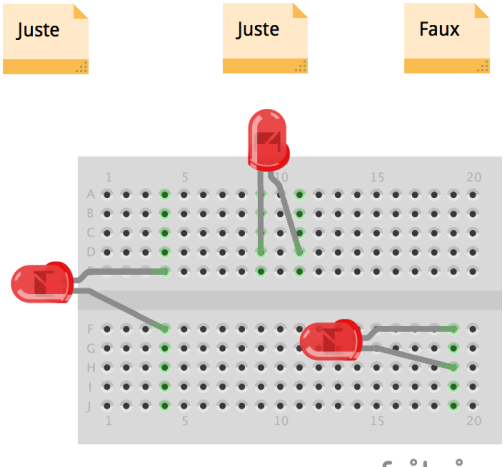
Une platine d'expérimentation (appelée *breadboard*) permet de réaliser des prototypes de montages électroniques sans soudure et donc de pouvoir réutiliser les composants.



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

Le logiciel Ardiuno fonctionne sur Mac, Windows et Linux. Il est nécessaire pour coder les programmes et les envoyer sur l'Arduino.

La communauté Arduino s'est séparée en deux branches en 2015. Depuis, chacune des branches développe son propre logiciel:

- La branche européenne exploite le site http://arduino.org et la marque Arduino
- La branche américaine exploite le site http://arduino.cc et les marques Arduino et Genuino

Les tribunaux décideront de la suite de cette affaire…

En attendant, les deux logiciels sont exploitable, quel que soit le module Arduino choisi. Chacune apporte un lot de fonctions que l'autre n'a pas.

- Télécharger le logiciel sur Arduino.org: http://www.arduino.org/downloads
- Télécharger le logiciel sur Arduino.cc: https://www.arduino.cc/en/Main/Software

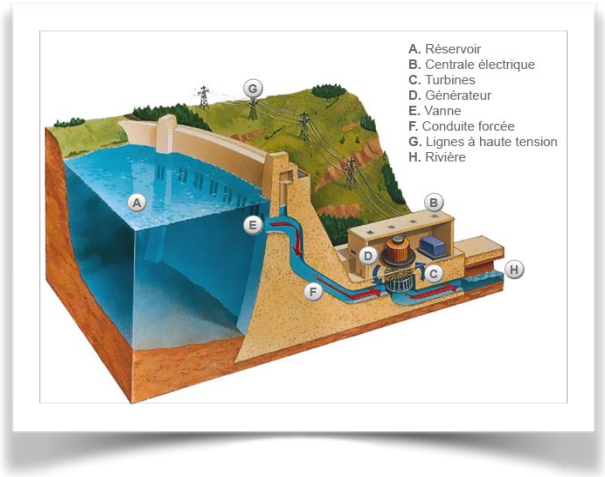
A noter que la version arduino.cc apporte une gestion améliorée des bibliothèques et sera plus adaptée pour des utilisateurs avancés.

Les bases de l'électronique

Petit rappel sur l'électricité

L'électricité est un déplacement d'électrons dans un milieu conducteur.

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)**.



On peut comparer le fonctionnement d'un circuit électrique à celui d'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 et de l'intensité I.

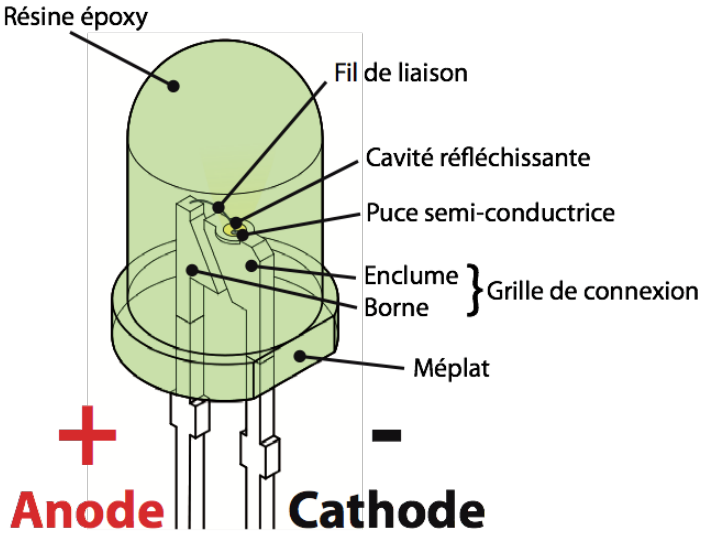
$$P = V \cdot I$$

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

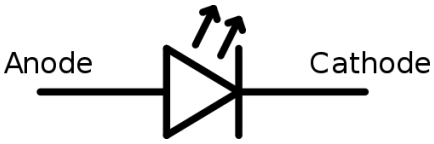
Les diodes

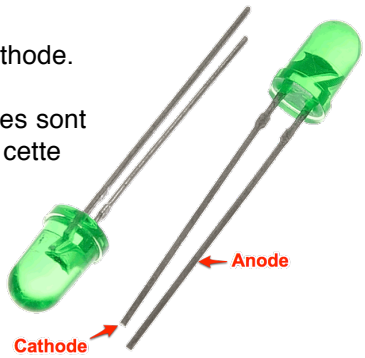
Il est possible de remplacer l'ampoule par une diode électroluminescente, aussi appelée LED[5]. **Elle a la particularité de ne laisser passer le courant électrique que dans un sens**.



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

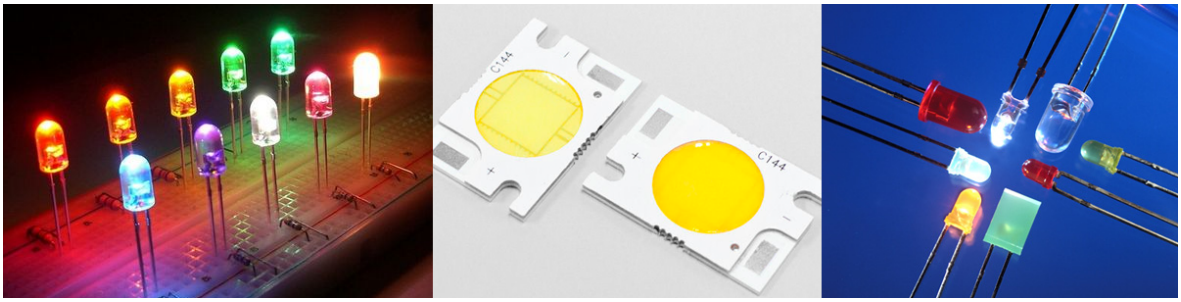
On reconnaît l'anode, car il s'agit de la broche la plus longue. Lorsque les deux broches sont de même longueur, on peut distinguer l'anode de la cathode, par un méplat du côté de cette dernière. Le symbole de la LED est le suivant:





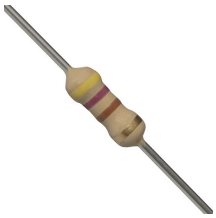
Attention: le courant produit par l'Arduino est trop important pour y brancher directement une LED dessus. **L'utilisation d'une résistance est obligatoire, pour ne pas endommager la LED**.

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.



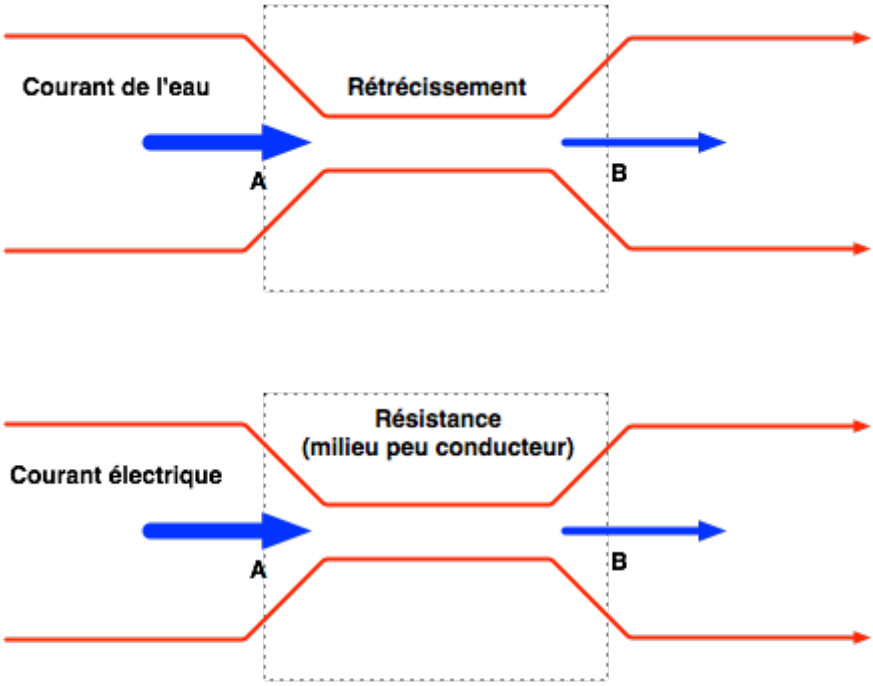
[5] http://fr.wikipedia.org/wiki/Diode_électroluminescente

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**.

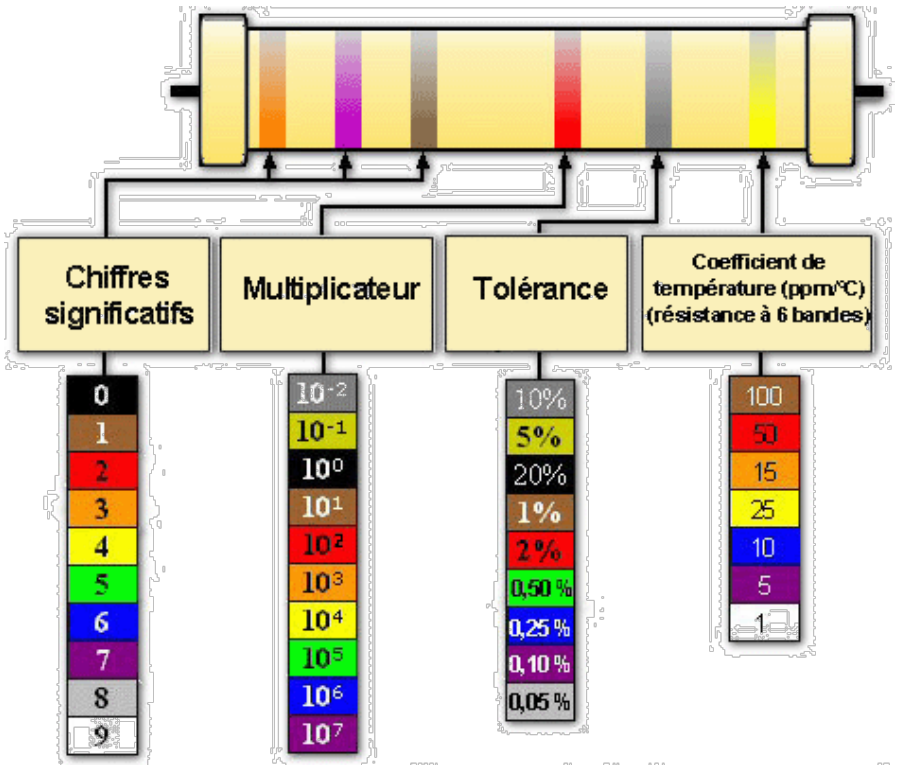


Ainsi, pour une tension fixe, plus la résistance est faible, plus le courant la traversant est fort. Cette proportion est vérifiée par la loi d'Ohm:

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

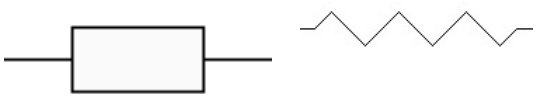
Une résistance est un milieu peu conducteur; les électrons peinent à s'y déplacer. Leur énergie se dissipe alors en général sous forme de chaleur. C'est ce principe utilisé pour les bouilloires électriques ou les ampoules à filaments.

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



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:



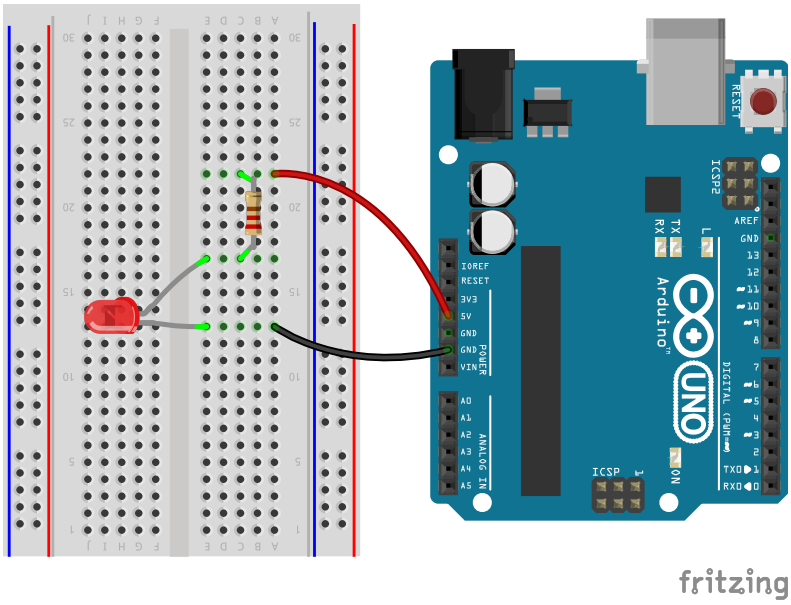
Un calculateur de valeurs de résistances est disponible à cette adresse: http://edurobot.ch/resistance/

Exercice1: 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.

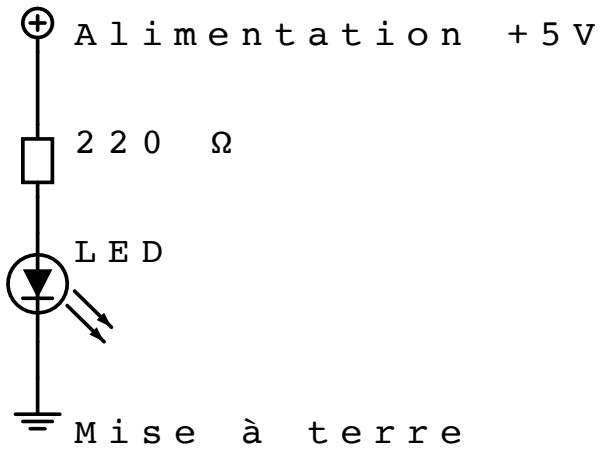
Circuit intro



Liste des composants:

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

Observations



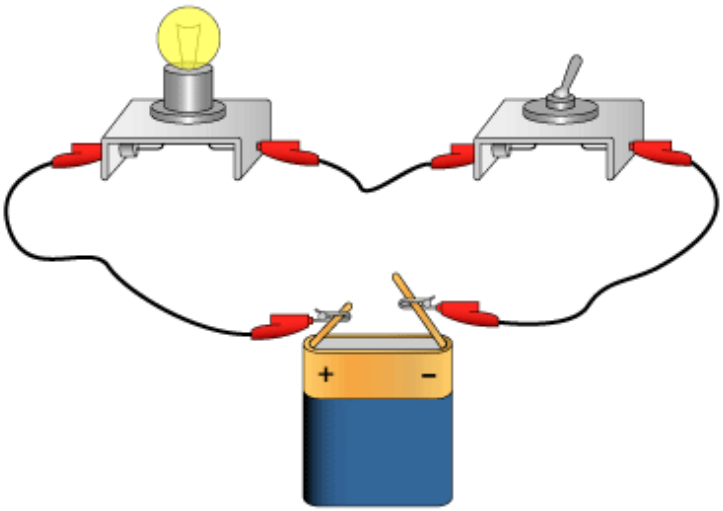
Voici ce que cela donne au niveau du schéma électronique.

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

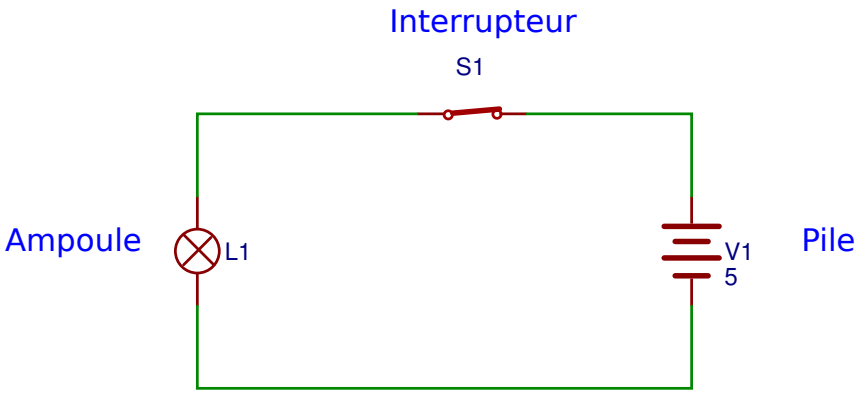
Lors de l'exercice 1, tu as réalisé un circuit électrique: tu as relié une source d'électron à la terre. Leur déplacement a ainsi permis à la LED de s'allumer. L'Arduino ne sert qu'à l'alimentation électrique, comme une pile.

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.

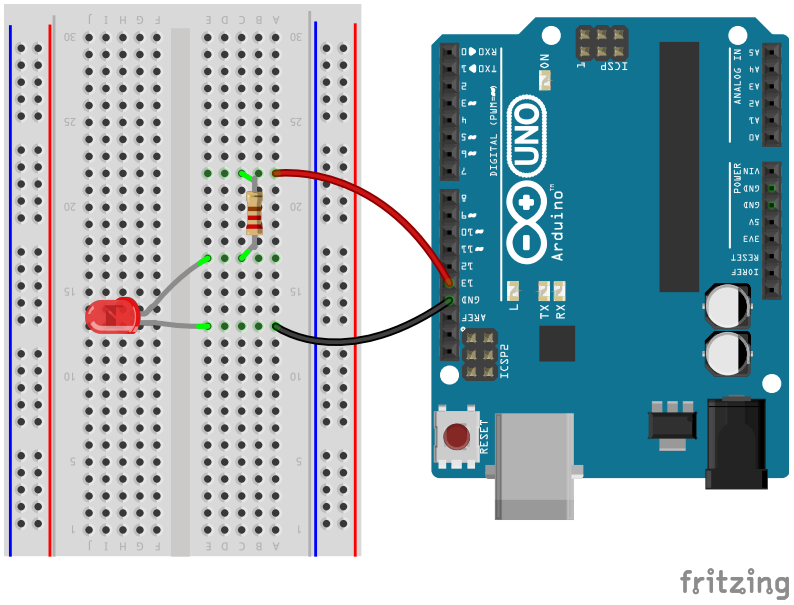
Lorsque l'interrupteur est déclenché, on dit que le circuit est **ouvert**. Le pôle positif n'étant alors plus relié au pôle négatif, les électrons ne peuvent plus se déplacer.

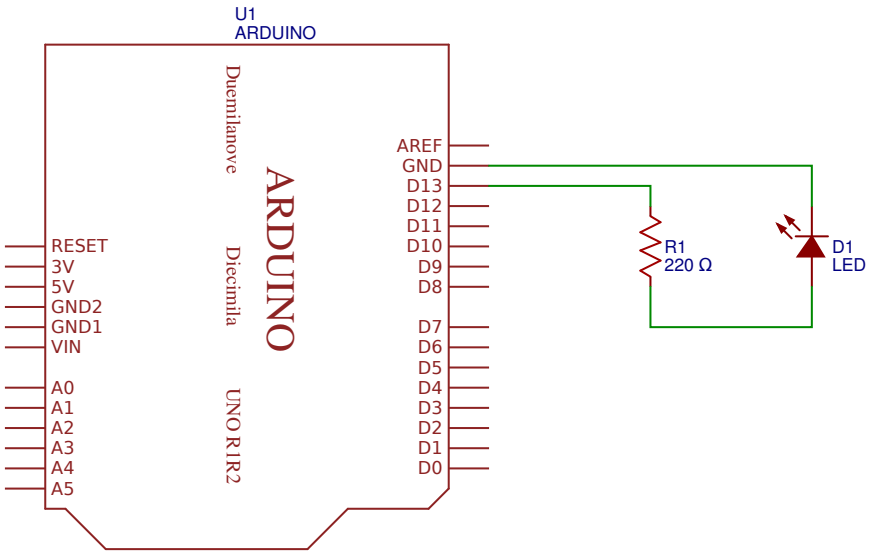
Exercice 2: faire clignoter une LED

Introduction

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

Circuit 1





Liste des composants:

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

La programmation se fait dans le logiciel Arduino IDE:

Le menu

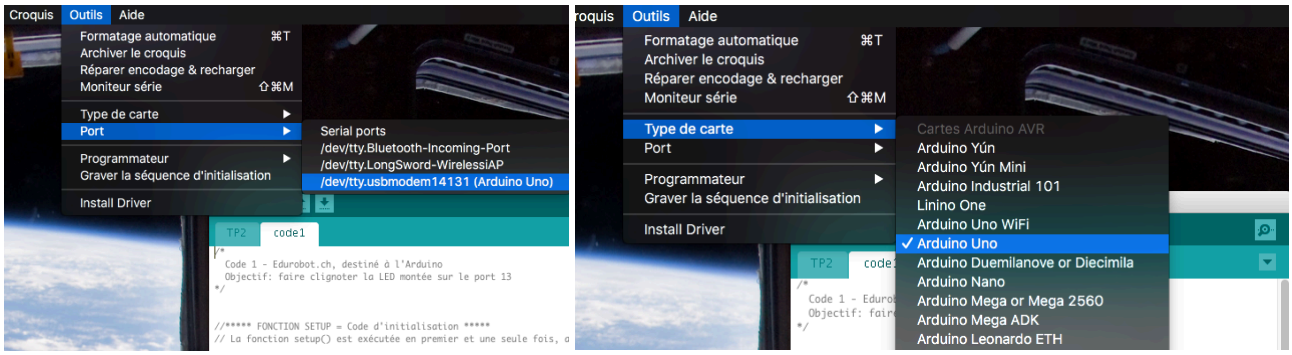
- Bouton 1 : Ce bouton permet de vérifier le programme, il actionne un module qui cherche les erreurs dans le programme
- Bouton 2 : Envoi du programme sur l'Arduino
- Bouton 3 : Créer un nouveau fichier
- Bouton 4 : Ouvrir un fichier existant
- Bouton 5 : Enregistrer un fichier

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

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 le port 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 setup()
```

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)*.

Observations

s

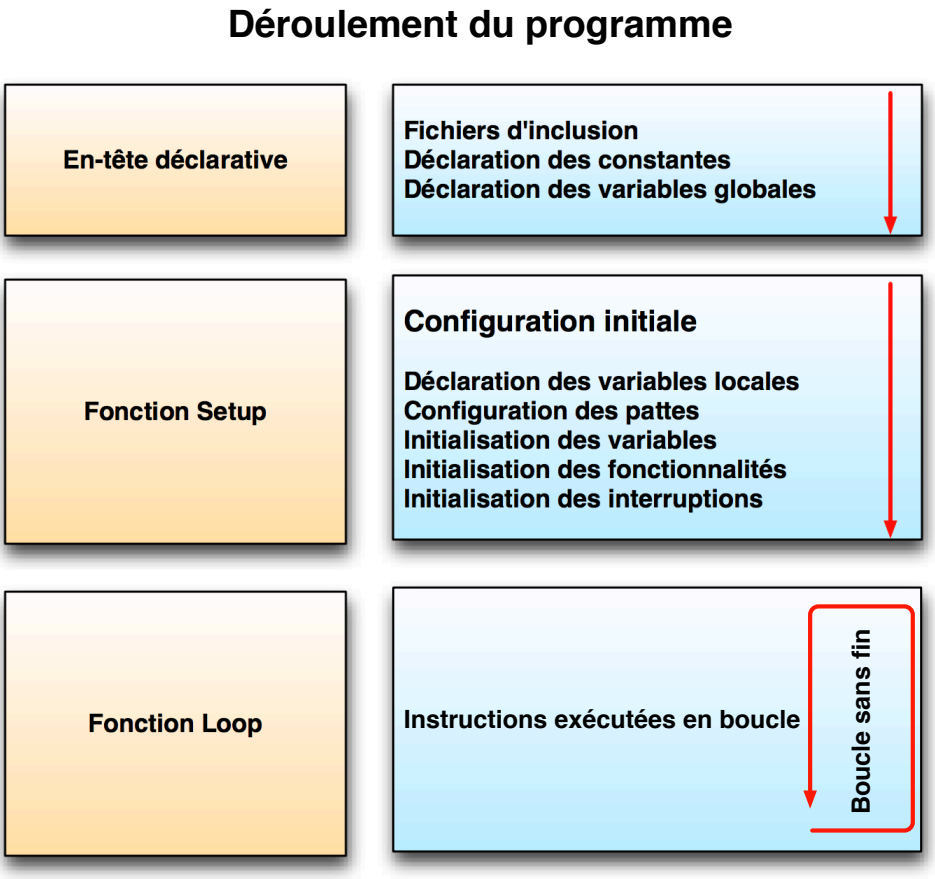
Ce code permet de faire clignoter la LED située sur la broche 13. Une LED, soudée sur l'Arduino et reliée à la broche 13 clignote elle aussi.

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 : "fait ceci, fait 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 :

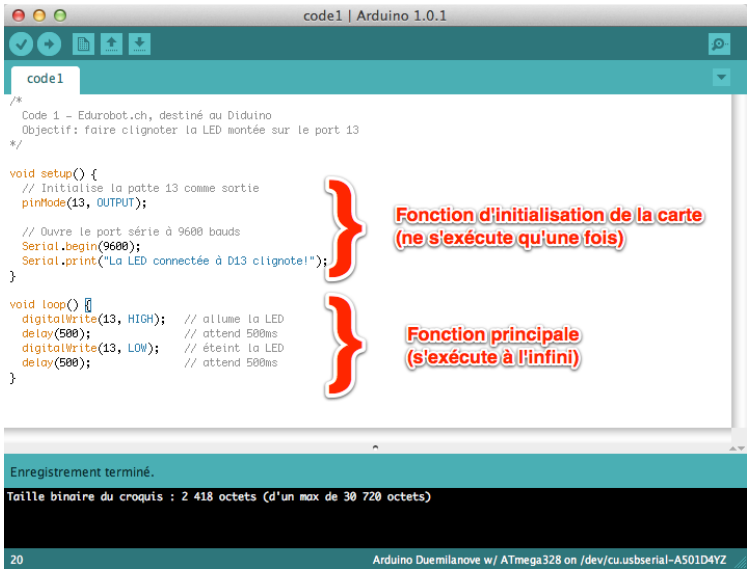
```
/*cette ligne est un commentaire, sur PLUSIEURS lignes
 qui sera ignoré par le programme, mais pas par celui qui lit le code ;) */
```

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 vaenvoyer 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'électicité 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);
```

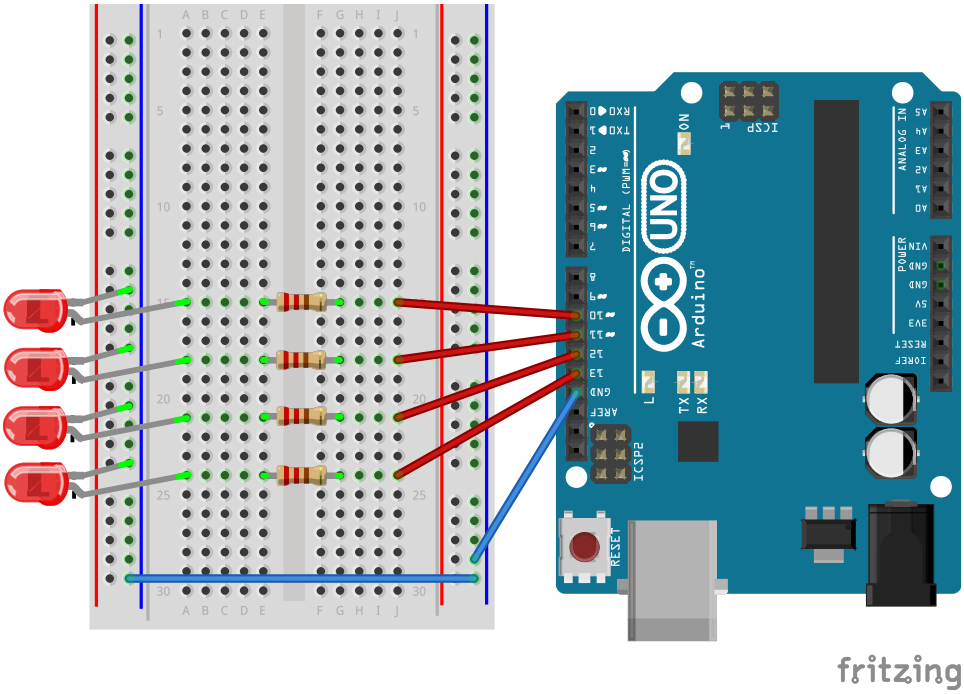
Quelle valeur de *delay* dois-tu choisir pour avoir l'impression que la LED cesse de clignoter?

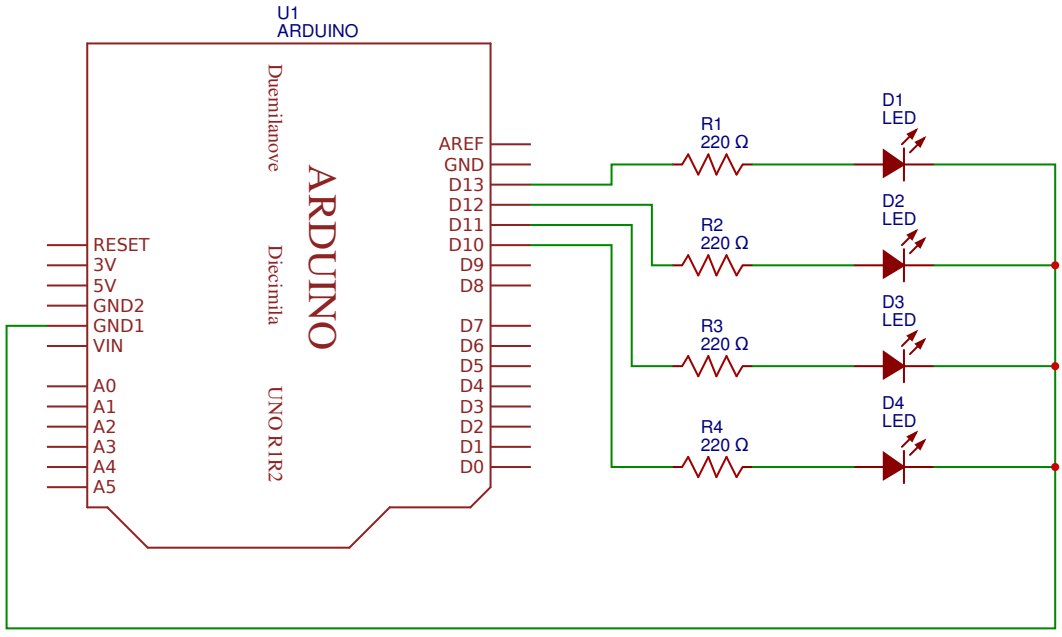
Réponse: .....

Exercice 3: faire clignoter quatre LEDs

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

Circuit 2





Liste des composants:

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

Code 2

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 ports 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 setup()
```

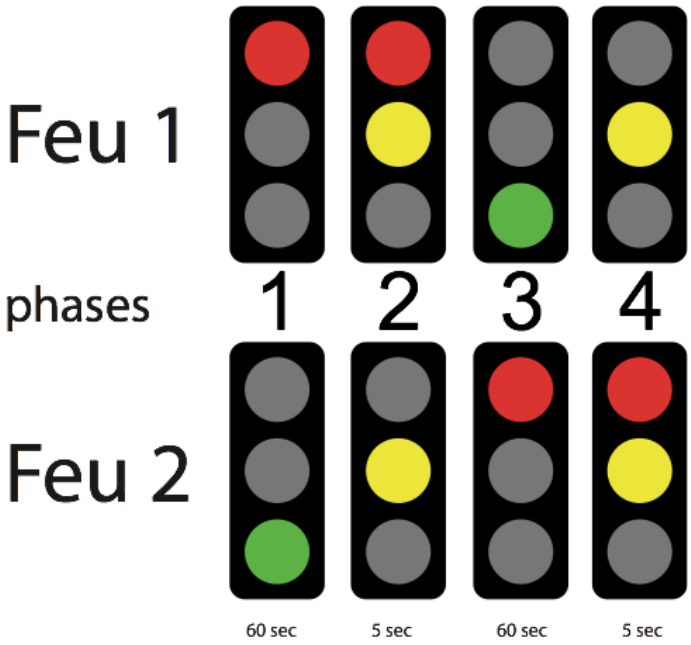
Code 3

```
/*
  Code 3 - Edurobot.ch, destiné à l'Arduino
  Objectif: faire un chenillard à 4 LEDs montées sur les ports 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, 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 setup()
```

Exercice 4: Les feux de circulation

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:



Tu peux visionner une vidéo de présentation ici: http://www.scolcast.ch/podcast/61/53-3067

Établis la liste des composants que tu auras besoin pour réaliser le projet:

Corrigé

Tu trouveras le corrigé à l'adresse suivante: http://edurobot.ch/code/code5.txt

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.

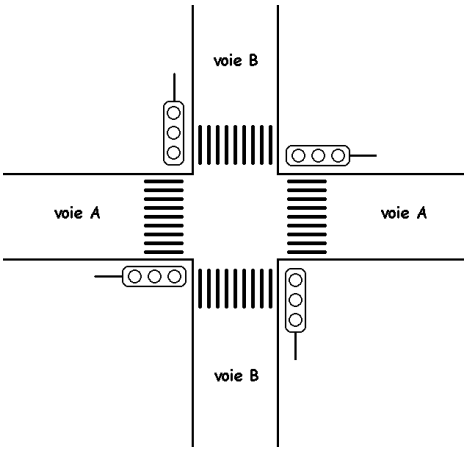
Variante 2

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:



Source: http://minne.romain.free.fr/MPI/tp10_carrefour/Gestion_d_un_carrefour.html

Exercice 4: les variables

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

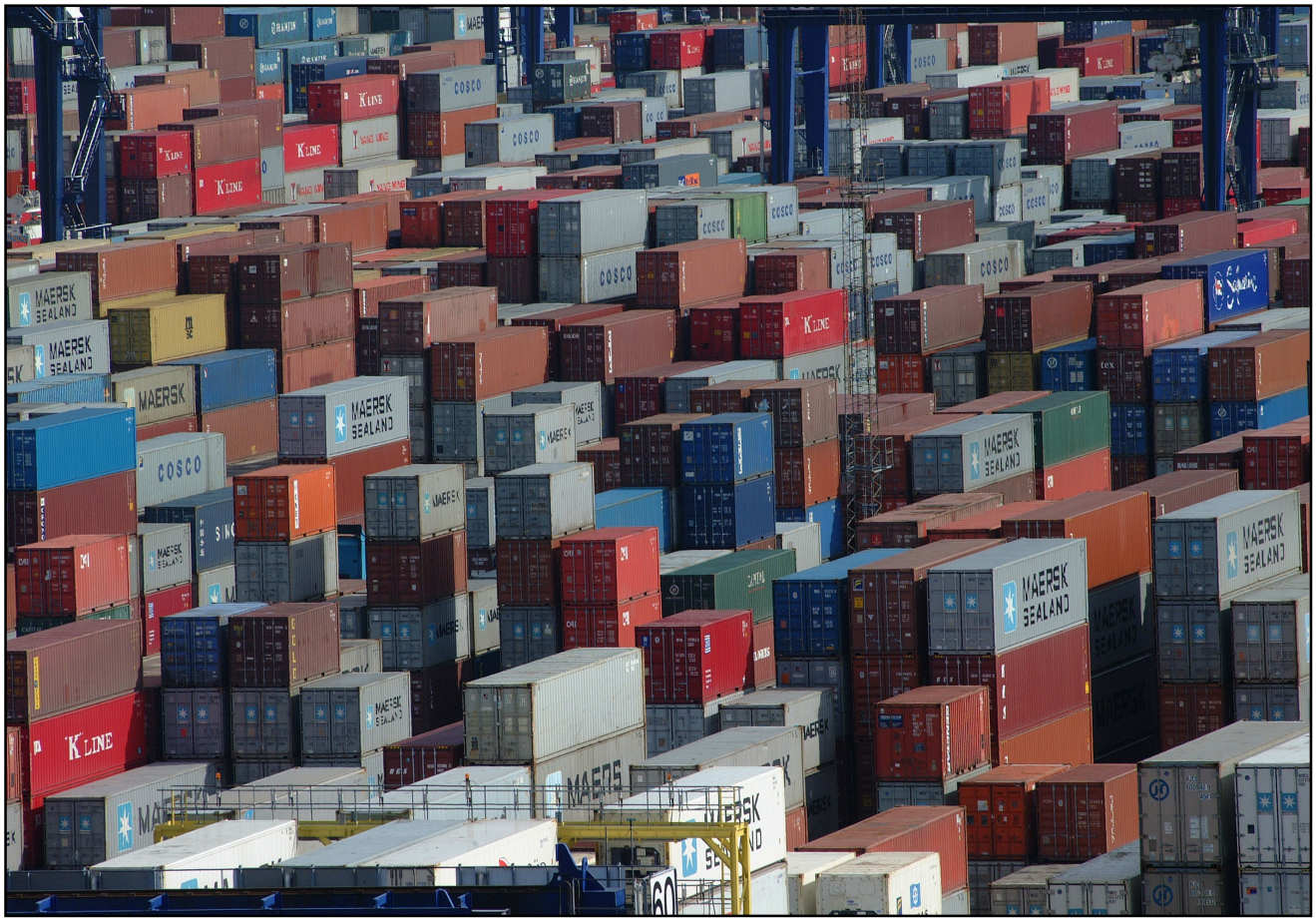
Imaginons un nombre dont nous devons nous souvenir. Ce nombre est stocké dans un espace de stockage de la mémoire vive (RAM) du microcontrôleur. Chaque espace de stockage est identifié de manière unique.

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 font qu'il faut 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.

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 chiffre 3.

Concrètement, qu'est-ce que cela signifie? Observons la différence entre les deux codes.

Broche non-définie	Broche 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);`
`}`
	`const int led_rouge = 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.

L'incrémentation

Il est possible d'appliquer à 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 ajoute 1 à la variable.

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

var++;

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

Code 4: faire clignoter 10 fois la LED 13

```
/*
Code 4 - 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 13 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[6]

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 stroque 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 prêt:

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

La déclaration *for* est habituellement utilisée pour répéter un bloc d'instructions entourées de parenthèses. On l'utilise habituellement 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:

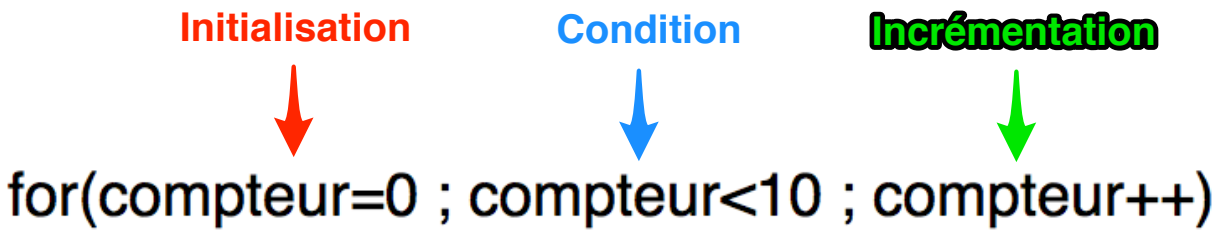
«Au commencement, la variable compteur est égale à zéro. On va exécuter le code en boucle. A chaque fois, on ajoute +1 à ta variable compteur, jusqu'à ce qu'on arrive à 9. A ce moment, on s'arrête.»

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

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(12, HIGH);   // allume la LED
    delay(500);               // attend 500ms
    digitalWrite(12, LOW);    // éteint la LED
    delay(500);               // attend 500ms
```

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

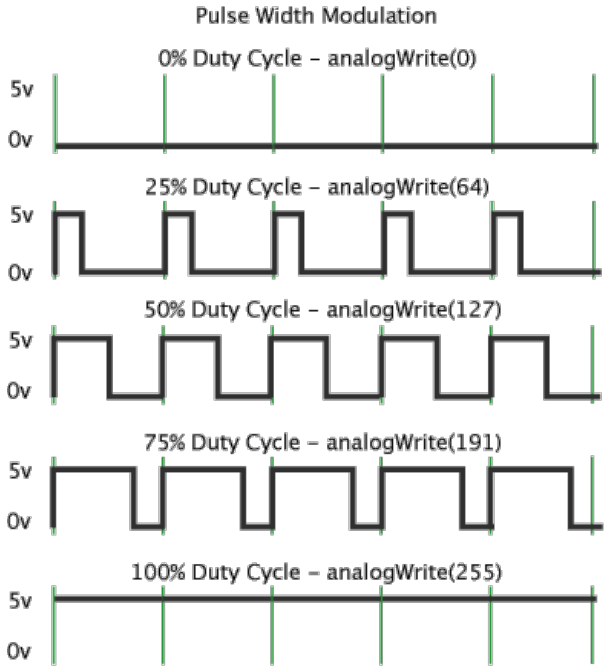
Nous pouvons sans problème utiliser une incrémentation et un *for* pour réaliser un chenillard:

```
/*
  Code 5 - 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 *****
// La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme


void setup() {

// Déclaration des broches 10 à 14 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 la 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 proche 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;
  }
}
```

Exercice 5: PWM

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 et basses 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%.



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:

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

Code 6: faire varier la luminosité d'une LED en modifiant la valeur PWM

```
/*
  Code 6 - 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 = 1000;   //On définit une durée de 1 seconde

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

```
// La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme


void setup() {

}

void loop() {

    analogWrite(ledPin, 0);
    // Attente de 30 millisecondes pour voir l'effet.
    delay(timer);

    analogWrite(ledPin, 50);
    // Attente de 30 millisecondes pour voir l'effet.
    delay(timer);

    analogWrite(ledPin, 100);
    // Attente de 30 millisecondes pour voir l'effet.
    delay(timer);

    analogWrite(ledPin, 150);
    // Attente de 30 millisecondes pour voir l'effet.
    delay(timer);

    analogWrite(ledPin, 200);
    // Attente de 30 millisecondes pour voir l'effet.
    delay(timer);

    analogWrite(ledPin, 255);
    // Attente de 30 millisecondes pour voir l'effet.
    delay(timer);
}
```

Code 7: faire varier la luminosité d'une LED en douceur

```
/*
  Code 7 - 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
*/


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


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


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

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


void setup() {

}

void loop() {
  // Variation du min au max par incrémentation de 5 pas sur 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 pour voir l'effet.
    delay(30);
  }

  // Variation du max au min par incrémentation de 5 pas sur 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 pour voir l'effet.
    delay(30);
  }
}
```

Exercice 6: les inputs

Jusqu'à maintenant, nous avons principalement des outputs, c'est-à-dire que de l'information sortait du microcontrôleur sous la forme d'un signal électrique *(HIGH)* ou de son absence *(LOW)* grâce aux commandes DigitalWrite et AnalogWrite. De même, il est possible d'envoyer un signal au microcontrôleur, depuis un capteur, par exemple. 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.

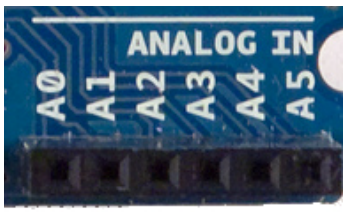
Les entrées analogiques

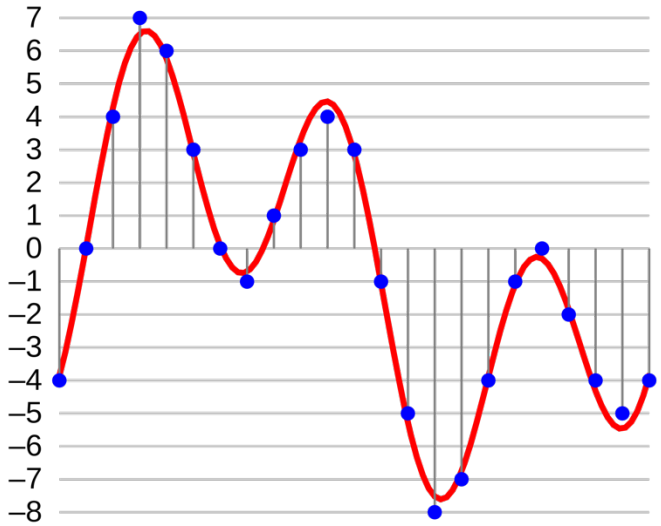
Un signal analogique[7] varie de façon continue au cours du temps. Sa valeur est donc un nombre réel. On trouve des signaux analogiques constamment, comme la température, la vitesse…

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.



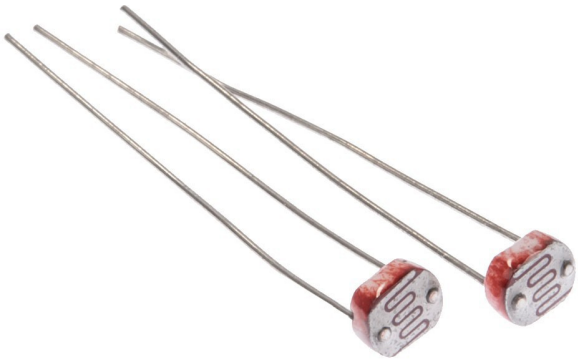


[7] Plus d'informations: http://f.hypotheses.org/wp-content/blogs.dir/904/files/2013/03/infographie_chloe_manceau.pdf

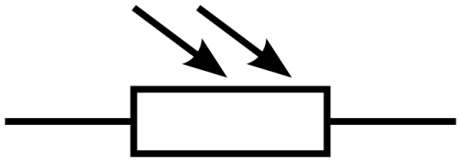
La photorésistance

Une photorésistance est un composant électronique dont la résistance varie en fonction de l'intensité lumineuse. Plus la luminosité est élevée, plus basse est la résistance. 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.
- …



Son symbole électronique est le suivant:



Pour le circuit 3, nous aurons besoin d'une photorésistance que nous connecterons à la broche A0.

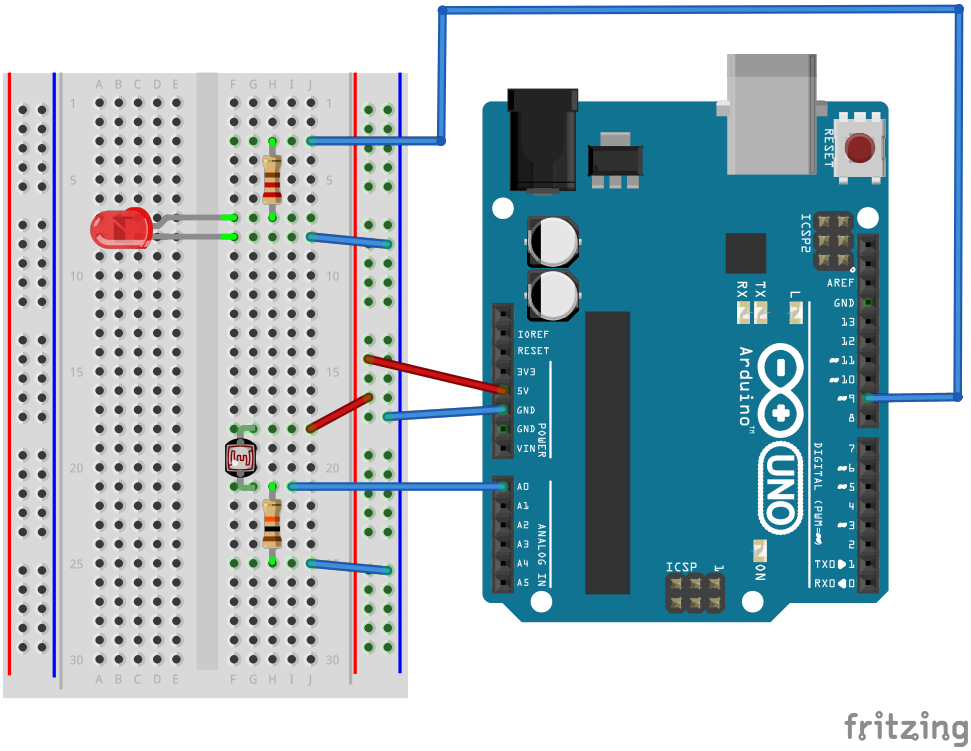
Circuit 3: diviseur de tension

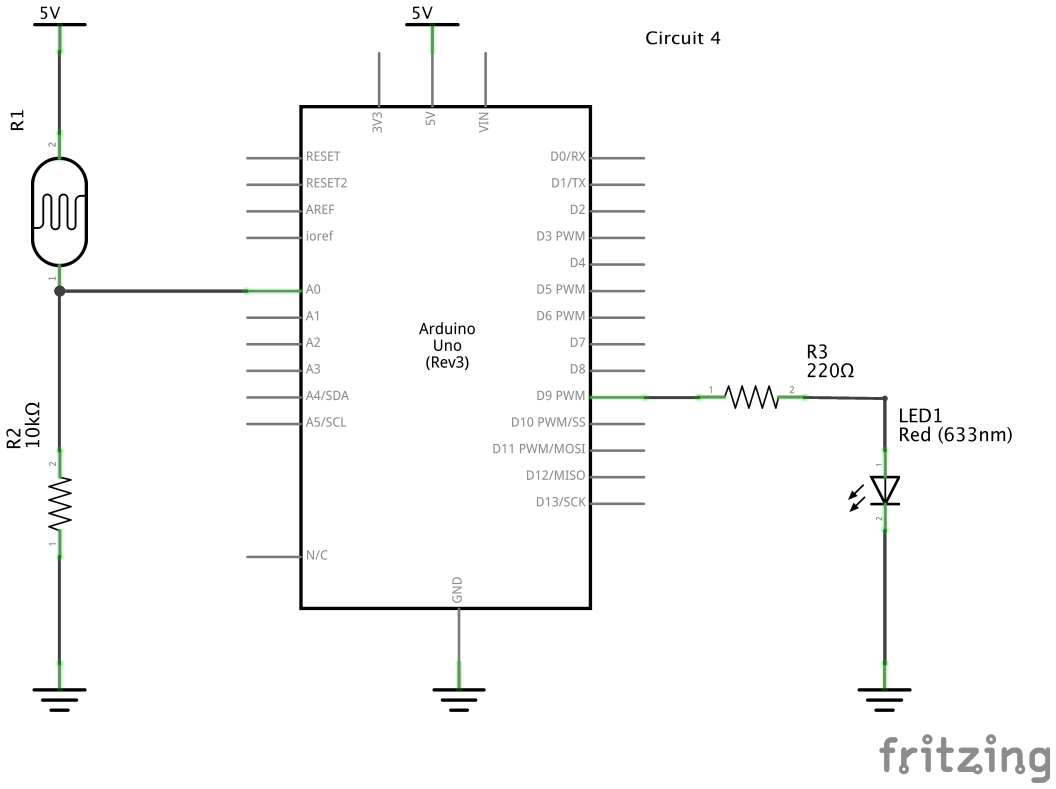
Le montage résistance fixe – photorésistance constitue ce qu'on appelle un **diviseur de tension** (5V qui vient d'Arduino).On relie le point entre les deux résistances à une broche analogique de l'Arduino et on mesure la tension par la fonction analogRead (broche). Tout changement de la tension mesurée 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 3


fritzing


Circuit 4
5V
5V
R1
R2
10kΩ
R3
220Ω
LED1
Red (633nm)
Arduino
Uno
(Rev3)
3V3
5V
VIN
RESET
RESET2
AREF
ioref
A0
A1
A2
A3
A4/SDA
A5/SCL
N/C
GND
D0/RX
D1/TX
D2
D3 PWM
D4
D5 PWM
D6 PWM
D7
D8
D9 PWM
D10 PWM/SS
D11 PWM/MOSI
D12/MISO
D13/SCK
fritzing

Code 8: 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*).

```
/*
  Code 8 - 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 ont éteint la LED.
     }
    else                                  //Sinon...
    {
     digitalWrite (ledPin, HIGH);         //...on allume la LED
    }
}
```