

Arduino à l'école

**Cours pour l'apprentissage des bases de
l'électronique et de la programmation sur Arduino**



Par Frédéric Genevey  - Jean-Pierre Dulex 

Date de publication : 9 décembre 2019

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. Il n'est pas fait pour être suivi de manière linéaire. Il sert en effet de cadre théorique. Après les bases acquises, l'enseignant a tout intérêt à travailler en projet en s'appuyant sur ce cours.

Commentez

I - Historique de ce cours.....	4
II - Consignes de sécurité.....	4
III - Préface.....	5
IV - Introduction.....	6
IV-A - Références.....	6
IV-B - Bibliographie.....	6
IV-C - Les meilleurs cours en ligne.....	11
IV-D - À propos des schémas électroniques.....	11
V - Matériel nécessaire.....	12
VI - Découverte de la plateforme Arduino.....	13
VI-A - Schéma d'une platine Arduino Uno.....	14
VI-B - Le microcontrôleur.....	14
VI-C - L'alimentation.....	15
VI-D - La connectique.....	15
VI-E - La platine d'expérimentation.....	16
VII - Le logiciel Arduino IDE.....	17
VIII - Les bases de l'électronique.....	19
VIII-A - Petit rappel sur l'électricité.....	19
VIII-B - Les diodes.....	20
VIII-C - Les résistances.....	21
IX - Projet 1 : le circuit électrique.....	24
IX-A - Le circuit électrique.....	25
X - Projet 2 : faire clignoter une LED.....	26
X-A - Le logiciel Arduino IDE.....	27
X-A-1 - Exercice : lancer un SOS.....	33
XI - Projet 3 : faire clignoter quatre LED.....	34
XII - Projet 4 : introduction aux variables.....	37
XII-A - Une variable, qu'est-ce que c'est ?.....	37
XII-B - Définir les broches du microcontrôleur.....	39
XIII - Projet 5 : les feux de circulation.....	40
XIII-A - Variantes.....	43
XIV - Projet 6 : l'incrémentation.....	44
XV - Projet 7 : PWM, variation en douceur d'une LED.....	48
XVI - Projet 8 : les inputs numériques.....	52
XVI-A - Protéger l'Arduino.....	52
XVI-B - Résistance Pull-Down / Pull-Up.....	53
XVI-C - Circuit 6 : montage avec résistance pull-down (rappel au moins).....	54
XVI-D - Petits exercices : bouton-poussoir et LED qui clignote.....	58
XVI-E - Le bargraphe.....	60
XVI-F - Déparasiter à l'aide de condensateurs.....	66
XVI-G - Variation : le bargraphe à 10 LED.....	67
XVI-H - Variation : l'afficheur numérique.....	71
XVI-I - Synthèse : apprendre à compter.....	79
XVI-J - Code 15 : apprendre à compter.....	81
XVII - Projet 9 : les inputs analogiques.....	85
XVII-A - La photorésistance.....	86
XVII-B - Circuit 11 : diviseur de tension.....	87
XVII-C - Code 16 : valeur de seuil.....	88
XVII-D - Code 17 : variation de la luminosité d'une LED en fonction de la lumière ambiante.....	89
XVII-E - Code 18 : mappage de données.....	91
XVIII - Projet 10 : le potentiomètre.....	92
XVIII-A - Circuit 12 : utiliser le potentiomètre pour faire varier la luminosité d'une LED.....	93
XVIII-B - Circuit 13 : utiliser le potentiomètre pour faire varier la luminosité d'une LED.....	95
XVIII-C - Circuit 14 : allumer les LED d'un bargraphe à l'aide d'un potentiomètre.....	96
XVIII-D - Code 20 : fficher la valeur d'un potentiomètre à l'aide d'un bargraphe.....	98
XIX - Projet 11 : construire une station météo.....	101
XIX-A - Code 21 : acquérir les données du capteur et les afficher.....	102
XIX-B - Code 22 : afficher les données sur l'écran LCD.....	106

XX - Projet 12 : utiliser un servomoteur.....	107
XX-A - Code 23 : faire bouger le bras d'un servomoteur dans les deux sens.....	111
XX-B - Code 24 : servomoteur et gestion des tâches.....	112
XX-C - Code 25 : commander un servomoteur avec un potentiomètre.....	113

I - Historique de ce cours

J'ai commencé ce cours en même temps que j'ai commencé à utiliser l'Arduino avec mes élèves en 2012. Cette année, on en a profité pour pirater le sapin de Noël du directeur de l'école (1). D'abord destiné à des élèves d'une option VSO de l'école vaudoise, ce cours est maintenant adapté à des élèves d'une OCOM MITIC ou Technologie, dès la 10H.

NDLR. Voir le système secondaire du canton de Vaud en Suisse.



10H : classe du 3^e cycle dans le système harmonisé, soit des élèves de 13-14 ans ! Ce qui explique le tutoiement dans le texte et certains codes... peu optimisés.

Vous pouvez envoyer vos remarques, vos idées et corrections à info@edurobot.ch.

Un merci particulier à Jean-Pierre Duxel pour la relecture et les tests réalisés avec ses élèves ainsi qu'au professeur Jean-Daniel Nicoud.

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

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



II - 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**.



Quelques règles élémentaires de sécurité

- Il ne faut jamais connecter directement l'Arduino au secteur (**230 Volts alternatif**).
- Pour l'alimentation des projets, utiliser des blocs d'alimentation avec adaptateur secteur répondant 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.
- Ce cours ne permet PAS d'acquérir les compétences et notions de sécurité nécessaires pour travailler avec le secteur (230 V), ni avec l'électricité automobile (12 V).

III - 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 (2) .

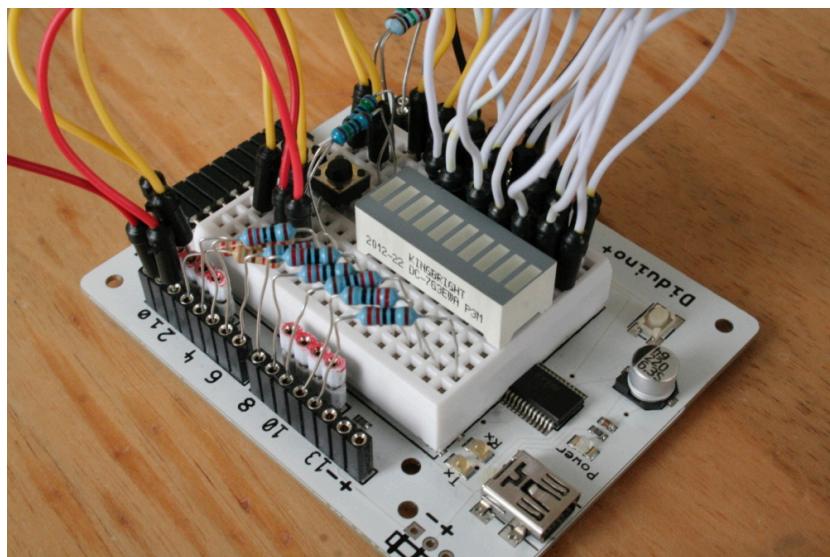
Ainsi, l'Arduino a été conçu dès le départ dans un but pédagogique, pour être bon marché, doté d'une grande quantité d'entrées et de sorties, compatible Mac, Windows et Linux, programmable avec un langage très simple et open source. Il n'y a là que des avantages pour le monde scolaire, en particulier parce que l'Arduino se situe au croisement entre l'informatique, l'électronique et les travaux manuels (3) .

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 1200 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.

i 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. Il n'est pas fait pour être suivi de manière linéaire. Il sert en effet de cadre théorique. Après les bases acquises, l'enseignant a tout intérêt à travailler en projet en s'appuyant sur ce cours.

Note : il n'y a pas de différence entre du matériel Arduino et Genuino. Faites attention avec les copies chinoises d'Arduino. Les moins chères peuvent poser des problèmes d'utilisation assez importants (drivers nécessaires, câblage avec faux contacts...).



Diduino, compatible Arduino, développé à Lausanne

IV - Introduction

IV-A - 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://mediawiki.e-apprendre.net/index.php/Diduino-Robot> ;
- <https://www.didel.com>.

Sources annexes :

- http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ARDUINO ;
- <http://makezine.com/category/technology/arduino/> ;
- <http://www.craslab.org/arduino/livrehtml/LivretArduinoCRAS.html> ;
- <http://arduino103.blogspot.ch> ;

Ce cours ne permet qu'une introduction à l'électronique. Par ailleurs, il faut noter l'excellent wiki suivant, qui propose de très nombreuses expériences sur Arduino. Idéal pour des TP ou pour donner des défis aux élèves :

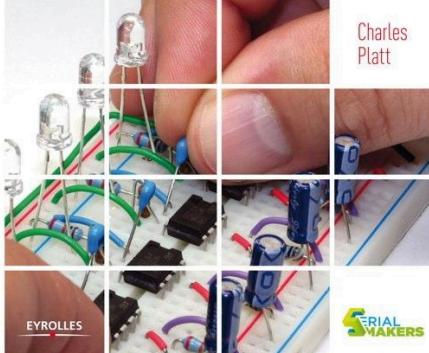
- www.wikidebrouillard.org/index.php?title=Catégorie:Arduino

IV-B - Bibliographie

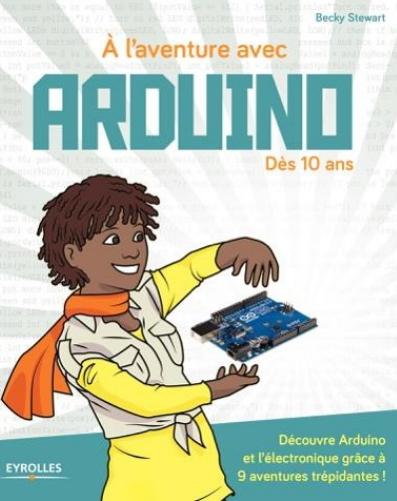
Il existe de nombreux livres sur Arduino et sur l'électronique. Voici une sélection de livres que nous avons évalués et sélectionnés.

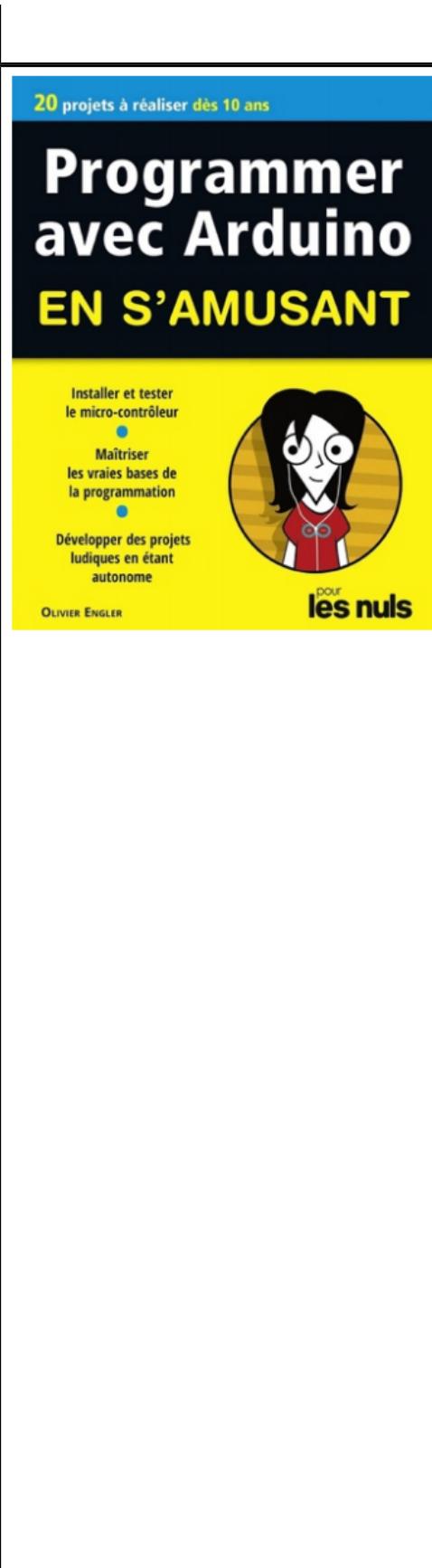
Note : nous vous encourageons à commander ces livres chez votre librairie. Mais si vous désirez les commander chez Amazon, nous vous serions reconnaissants d'utiliser les liens sponsorisés ci-dessous. Ils me permettent de gagner de quoi commander un ou deux livres Arduino par année.

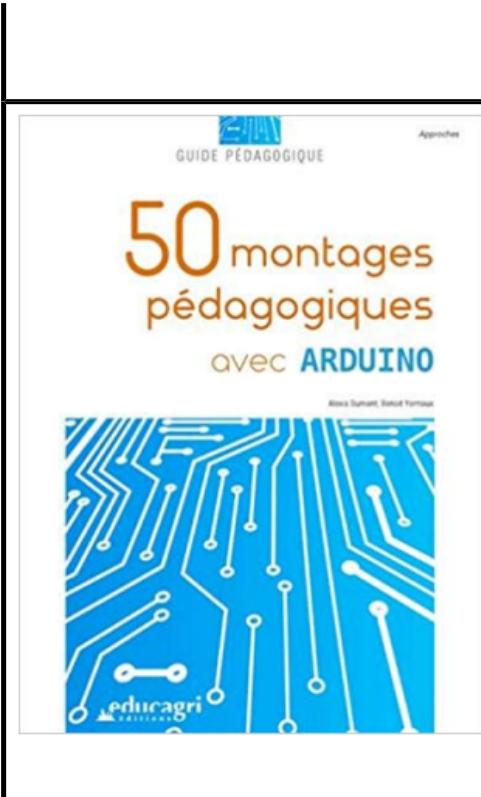
Électronique

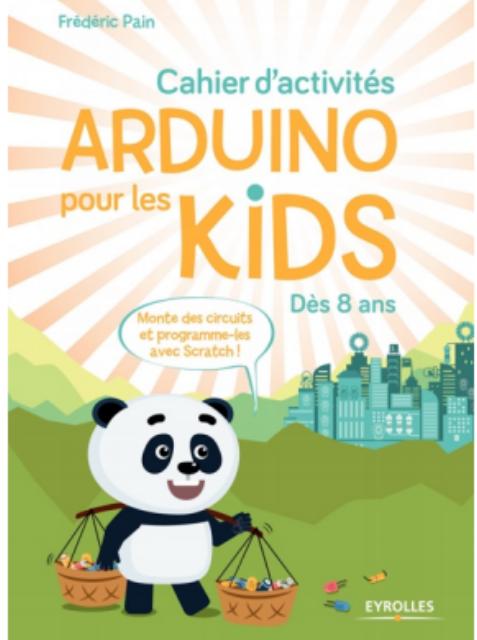
 <p>L'ÉLECTRONIQUE EN PRATIQUE 36 expériences ludiques</p> <p>Charles Platt</p> <p>EYROLLES</p> <p>SERIAL MAKERS</p>	<p>L'électronique en pratique, de Charles Platt ISBN: 978-2212135077</p> <p>L'approche pédagogique de ce livre est l'apprentissage par la pratique. On commence par expérimenter et découvrir, et ensuite seulement vient la théorie pour affirmer 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. Il s'agit donc d'un excellent livre pour l'enseignant.</p>
 <p>Øyvind Nydal Dahl</p> <p>L'ÉLECTRONIQUE pour les KIDS Dès 10 ans</p> <p>EYROLLES</p>	<p>L'électronique pour les Kids, de Øyvind Nydal Dahl ISBN: 978-2212118629</p> <p>Il est parfois difficile d'aborder et de faire comprendre les notions d'électricité et d'électronique aux élèves. C'est là que le livre <i>L'électronique pour les kids</i> vient apporter une approche intéressante : on ne s'embarrasse pas d'Arduino et de programmation. L'entier du premier tiers du livre, soit la partie 1, est consacré à l'électricité. De manière simple et pédagogique, l'enfant comprend ce qu'est l'électricité. La seconde partie permet la découverte des différents composants électroniques au travers de petits circuits, dont certains sont soudés. Enfin, la troisième partie est une approche du monde numérique, mais par l'électronique. Et c'est cette partie, elle est géniale ! On aborde les circuits logiques, le binaire...</p>

Arduino

 <p>LE GRAND LIVRE D'ARDUINO</p> <p>Erik Bartmann</p> <p>EYROLLES</p> <p>MAKERS</p>	<p>Le grand livre d'Arduino, d'Erik Bartmann ISBN: 978-2212137019</p> <p>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.</p>
 <p>Démarrez avec Arduino</p> <p>Principes de base et premiers montages</p> <p>2^e édition</p> <p>Massimo Banzi</p> <p>Co inventeur d'Arduino</p> <p>COLLECTION ETSF</p> <p>DUNOD</p>	<p>Démarrez avec Arduino – 2^e édition : Principes de base et premiers montages, par Massimo Banzi ISBN: 978-2100701520</p> <p>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. À conseiller à tous les débutants.</p>
 <p>À l'aventure avec ARDUINO</p> <p>Dès 10 ans</p> <p>Becky Stewart</p> <p>Découvre Arduino et l'électronique grâce à 9 aventures trépidantes !</p> <p>EYROLLES</p>	<p>À l'aventure avec Arduino – Dès 10 ans, par Becky Stewart ISBN: 978-2212143140</p> <p>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</p>

	<p>aborder les registres à décalage et les servomoteurs.</p> <p>Programmer avec Arduino en s'amusant pour les nuls, par Olivier Engler</p> <p>ISBN: 978-2412023877</p> <p>Tout le monde connaît la série de livres Pour les nuls. Le nom de la collection est trompeur ; chaque livre va étudier un sujet à fond, mais il va prendre par la main le lecteur, étape après étape. Il est indiqué sur la couverture : <i>20 projets à réaliser dès 10 ans</i>.</p> <p>Alors ? Eh bien, on ne s'amuse pas, et à 10 ans, on aura mieux à faire que de se plonger dans ce livre. Et pourtant, cela reste l'un des meilleurs livres pour découvrir le monde d'Arduino, mais à destination des ados passionnés ou des adultes qui n'ont aucune notion de programmation ou d'Arduino et qui sont intéressés par découvrir ce monde... ou accompagner un jeune. De fait, c'est un excellent livre pour tout enseignant(e) n'ayant pas des années de programmation derrière lui/elle et qui désire embarquer ses élèves dans l'aventure Arduino. À condition d'accepter le tutoiement à la mode Ikea... et les marottes linguistiques de l'auteur, qui s'évertue à appeler <i>mikon</i> un microcontrôleur.</p> <p>Le premier montage électronique n'intervient que juste avant la page 100. Avant cela, il n'y a que peu de pratique. On aura perdu tous les jeunes bien avant ça. Mais alors, que se passe-t-il durant les 100 premières pages ? Eh bien une présentation, claire, didactique, de l'Arduino, comme je n'en ai jamais vu dans d'autres livres. On aborde même le langage assembleur (brièvement, mais on en parle, avec même un exemple). C'est aussi le premier livre qui présente simplement et clairement ce qu'est un microcontrôleur et à quoi il sert.</p> <p>Chaque élément de la carte Arduino Uno est ainsi présenté. C'est complet, simple, compréhensible. Chaque concept est ainsi présenté. Normalement, dans un cours Arduino, on commence par faire des montages sur les broches numériques. Là pas, on commence avec les broches analogiques.</p>
--	--

	<p>Étonnant. Rafraîchissant. Et pas si bête. On comprend tous une variation d'une valeur analogique.</p> <p>50 montages pédagogiques avec Arduino, par Alexis Dumont et Benoit Yernaux ISBN: 979-1027501441 Voici enfin un livre explicitement destiné aux enseignants. Ici, point de théorie, mais immédiatement de la pratique. Comme son titre l'indique, le livre est un catalogue de projets Arduino, toujours plus complexes et destinés à la classe. Il ne s'agit pas ici de ne travailler que la programmation, mais aussi d'utiliser Arduino en mathématiques, physique, biologie... Une bonne base pour des travaux pratiques ou des projets d'élèves. Ce livre n'est pas destiné à des débutants dans le monde Arduino, mais bien à l'enseignant, qui ayant déjà de bonnes bases, cherche des projets pour ses élèves. Le lien avec les différentes disciplines scolaires est le vrai plus de ce livre.</p>
	<p>Sylvia présente : super projets Arduino, par Sylvia Todd ISBN: 978-2412023891 Il n'est pas commun qu'un livre sur Arduino soit écrit par une jeune demoiselle. Mais ce n'est pas n'importe quelle demoiselle ; elle est déjà une figure reconnue dans le mouvement maker. Ce petit livre, très coloré, superbement illustré, s'adresse clairement aux enfants, mais les adultes débutants sur Arduino y trouveront tout autant leur compte. Comme le titre l'indique, il ne s'agit pas d'apprendre à programmer sur Arduino, mais d'approcher ce monde avec de</p>

	<p>petits projets qui, mine de rien, vont parfois assez loin. Une réussite.</p> <p>Cahier d'activités Arduino pour les kids : Dès 8 ans, par Frédéric Pain ISBN: 978-2212675696</p> <p>Il est aussi possible de programmer Arduino à l'aide de Scratch et du logiciel mBlock. Ce livre est très clair et bien illustré. Il nous guide au travers de sept montages pour aboutir à la création d'une minestation météo. La programmation se fait via Scratch, donc sans code et le tout en français. Idéal donc pour permettre aux jeunes curieux de découvrir le monde Arduino.</p> <p>Je reste néanmoins dubitatif par le montage « pluviomètre ». Avec mon expérience, j'essaierai d'éviter de faire cohabiter de l'eau avec de l'électronique et un ordinateur...</p>
---	---

IV-C - Les meilleurs cours en ligne

- <http://www.instructables.com/class/Arduino-Class> (anglais)



NDLR. Sans oublier **les meilleurs cours et tutoriels pour utiliser la carte Arduino sur Developpez.**

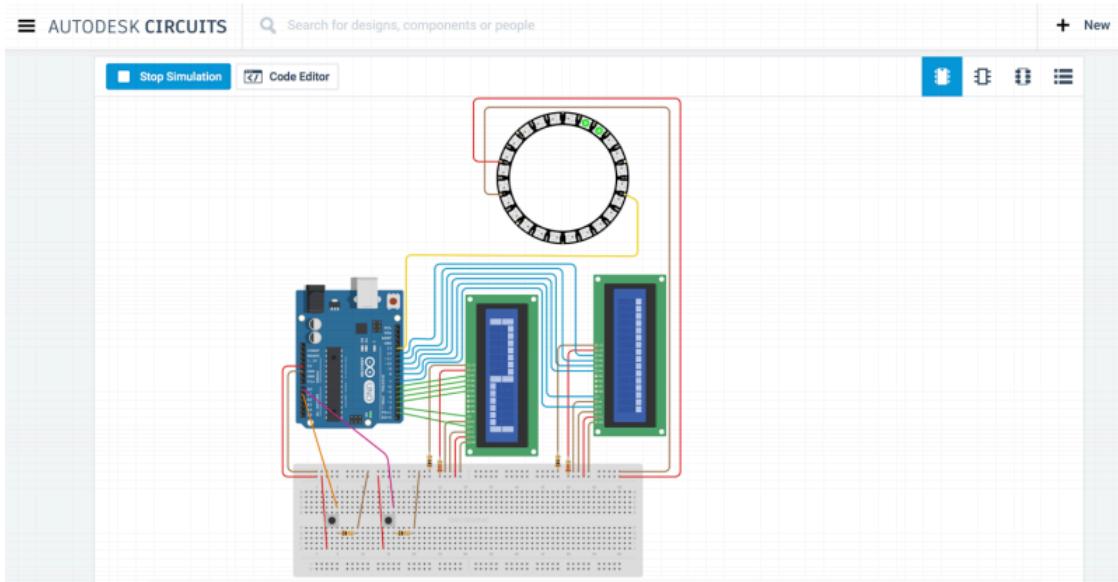
IV-D - À propos des schémas électroniques

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

- Fritzing : <http://fritzing.org> ;
- Solve Elec : <http://www.physicsbox.com/indexsolveelec2fr.html> ;
- Fido Cadj : <http://davbucci.chez-alice.fr/index.php?argument=elettronica/fidocadj/fidocadj.inc> ;
- QelectroTech : <https://qelectrotech.org> ;
- iCircuit : <http://icircuitapp.com>.

Il existe aussi une solution très élégante en ligne (4) : <https://easyeda.com>. Ce site permet de créer des schémas électroniques ainsi que la réalisation de circuits électroniques.

Un simulateur Arduino est aussi disponible à cette adresse : <https://www.tinkercad.com>. Il permet la réalisation de schémas d'implantation des composants sur les platines d'expérimentation, tout comme Fritzing, mais aussi la réalisation de schémas et de circuits électroniques.



V - Matériel nécessaire

Voici la liste minimum du matériel nécessaire pour suivre ce cours :

- une carte Arduino ou compatible Arduino ;
- une platine d'expérimentation (*breadboard*) ;
- des câbles de liaison (*jumpers*) ;
- 6 LED rouges ;
- 2 LED vertes ;
- 2 LED jaunes ou orange ;
- 1 LED RGB anode ou cathode commune ;
- 10 résistances de 220 à 470 Ω ;
- 2 résistances de 1 à 10 k Ω ;
- 2 condensateurs 10 nF ;
- 2 boutons-poussoirs ;
- 1 photorésistance ;
- 1 bargraphe 10 LED* ;
- 1 multimètre ;
- 1 potentiomètre (résistance variable) de 10 ou 20 k Ω ;
- 1 servomoteur ;
- 1 joystick*.

Pour pouvoir aller plus loin, voici les composants nécessaires :

- capteur de température et d'humidité DHT11 ;
- 1 écran LED 2x12 ;

(*optionnel)

Note : cette liste évoluera en même temps que le cours.

VI - Découverte de la plateforme Arduino



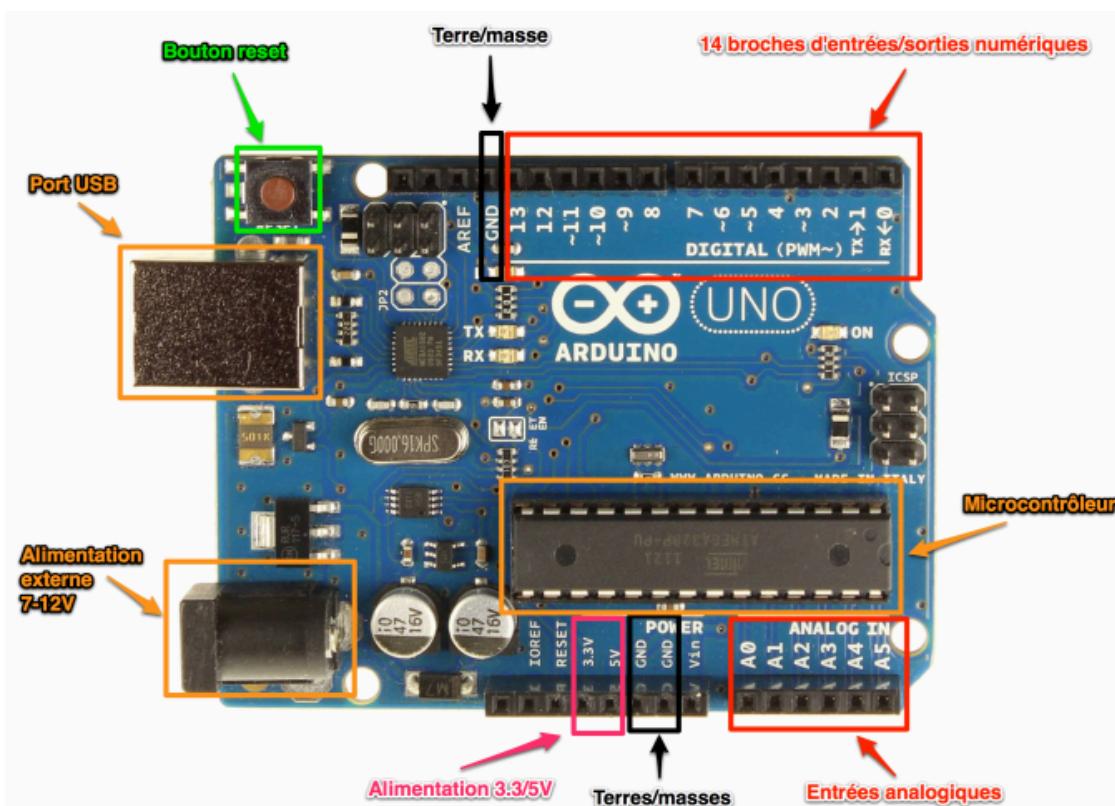
Arduino fait partie de la famille des platines de développement. Contrairement aux Raspberry Pi et aux Beaglebone, il ne possède pas d'OS fondé sur Linux. Un ordinateur est donc nécessaire. Il reste par contre l'un des plus abordables et des plus répandus. Une platine de développement est en général un circuit imprimé équipé d'un microprocesseur ou d'un microcontrôleur. Comme Arduino est open source, il existe un grand nombre de clones et de platines compatibles, tout comme il existe de nombreux modèles d'Arduino officiels, avec des fonctions particulières :



Pour ce cours, nous nous baserons sur le plus connu : l'Arduino Uno. Mais un autre modèle ou un clone fera aussi bien l'affaire. Vous pouvez par exemple trouver le Diduino, réalisé spécialement pour l'éducation ici :

- <https://www.didel.com/product-category/carte/cardui/>

VI-A - Schéma d'une platine Arduino Uno



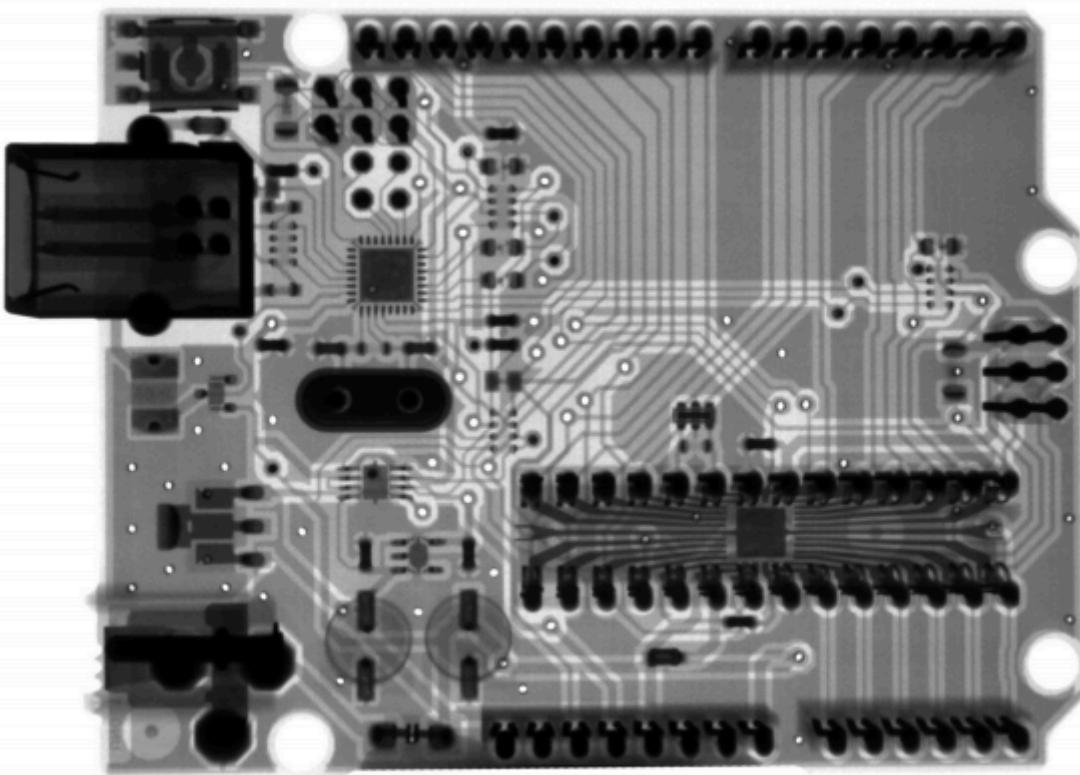
VI-B - Le microcontrôleur

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.



Lorsqu'on observe une carte Arduino Uno aux rayons X, on constate que la grande puce contient en réalité un microcontrôleur SMD. Noter aussi les pistes qui relient les broches au microcontrôleur :



Source : Mathew Schwartz

VI-C - L'alimentation

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



Attention : les cartes Arduino Due ainsi que d'autres cartes récentes fonctionnent avec une tension de 3.3 V au niveau des sorties ! La tension d'alimentation est similaire à l'Arduino Uno. Dans ce cours, nous partons du principe que la tension de fonctionnement des montages est de 5 Volts.

VI-D - La connectique

À 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).

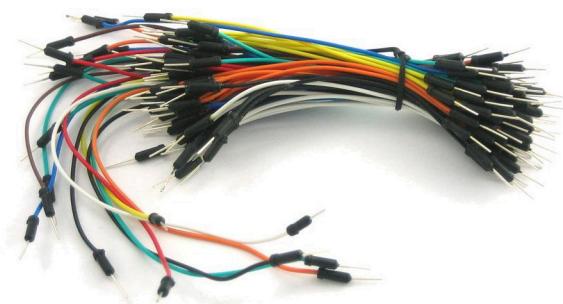
Sur les Arduino et sur beaucoup de cartes compatibles Arduino, les broches se trouvent au même endroit. Cela permet de fixer des cartes d'extension, appelée *shields* en les empilant.

Exploration des broches Arduino



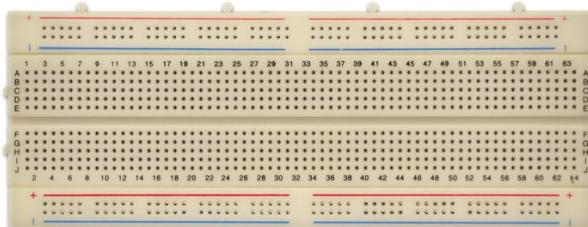
- **0 à 13** Entrées/sorties numériques ;
- **A0 à A5** Entrées/sorties analogiques ;
- **GND** Terre ou masse (0 V) ;
- **5 V** Alimentation +5 V ;
- **3.3 V** Alimentation +3.3 V ;
- **Vin** Alimentation non stabilisée (= le même voltage que celui à l'entrée de la carte).

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

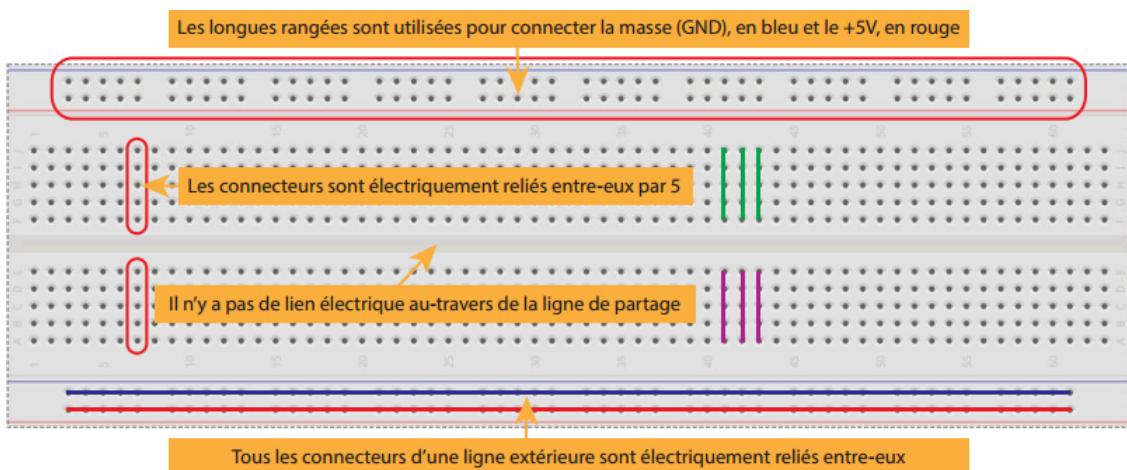


VI-E - La platine d'expérimentation

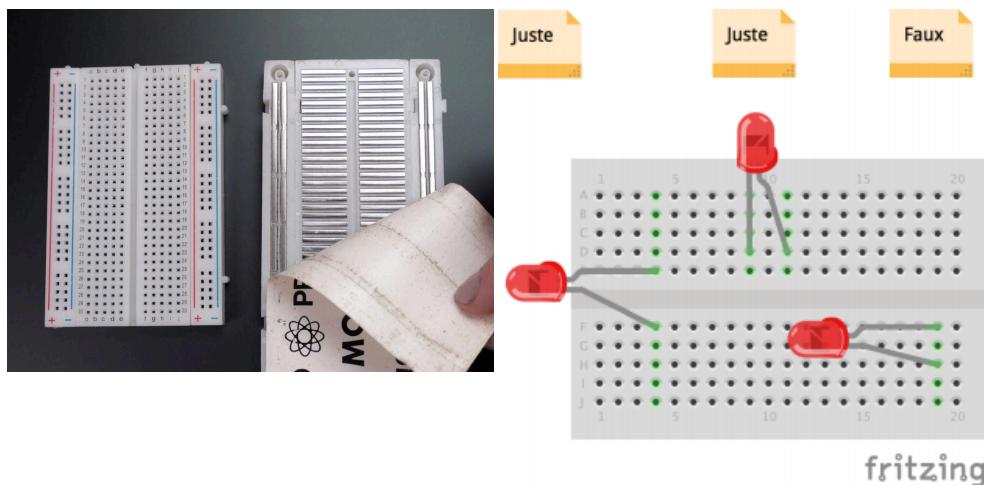
Une platine d'expérimentation (appelée *breadboard*) permet de réaliser des prototypes de montages électroniques sans soudure et donc de pouvoir réutiliser les composants.



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.



fritzing

VII - 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>.

TP2 | Arduino 1.6.11

```

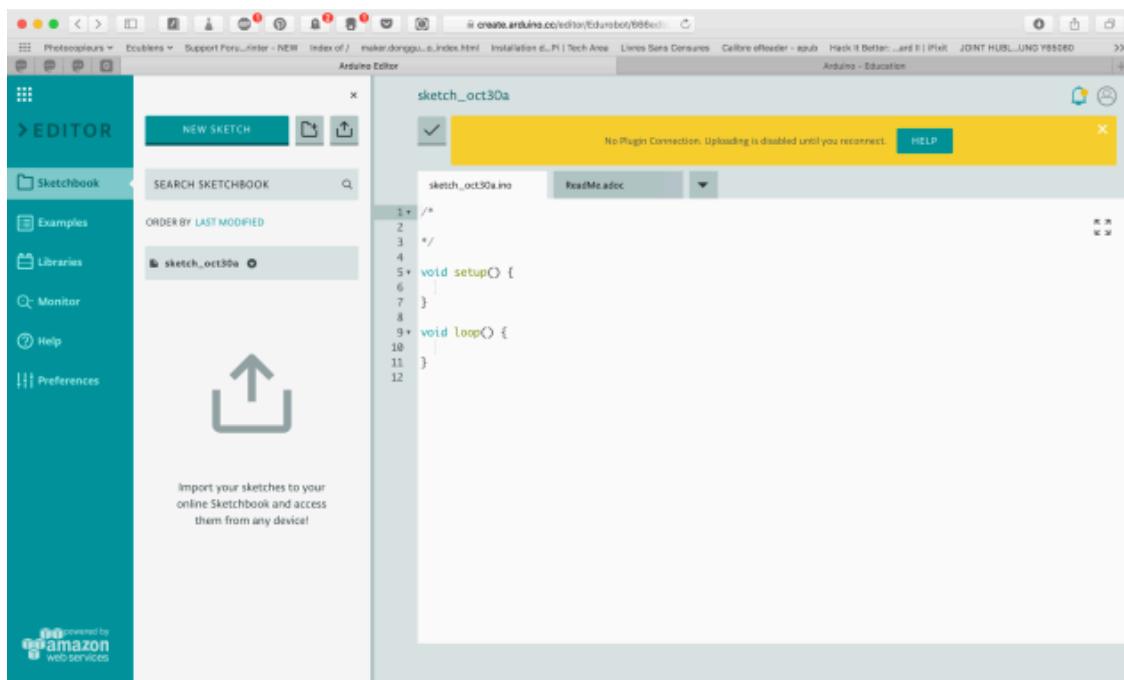
TP2
//rouge piéton rouge, memoire, //feu piéton du rouge
delay(3000); //durée 3 secondes

buttonState = digitalRead(buttonPin); //lecture de l'état du bouton
if ((buttonState != memoire) && (buttonState == HIGH)) //Comparaison de l'état du bouton par
//si l'état du bouton est différent
//stocké dans "memoire" (=LOW), alors
{
    digitalWrite(verte, LOW); //feu vert éteint
    digitalWrite(orange, HIGH); //feu orange allumé
    delay(1000); //durée 1 seconde
    digitalWrite(orange, LOW); //feu orange éteint
    digitalWrite(rouge, HIGH); //feu rouge allumé
    digitalWrite(p_vert, HIGH); //feu piéton vert allumé
    digitalWrite(p_rouge, LOW); //feu piéton rouge éteint
    delay(5000); //durée 5 secondes
    digitalWrite(p_vert, LOW); //feu piéton vert éteint
}
else //sinon on exécute le code suivant, soit le cycle des feux par
{
    digitalWrite(verte, LOW); //feu vert éteint
    digitalWrite(orange, HIGH); //feu orange allumé
    delay(1000); //durée 1 seconde
    digitalWrite(orange, LOW); //feu orange éteint
    digitalWrite(rouge, HIGH); //feu rouge allumé
    delay(3000); //durée 3 secondes
}

```

5 Arduino/Genuino Uno sur /dev/tty.usbmodem1421

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



VIII - Les bases de l'électronique

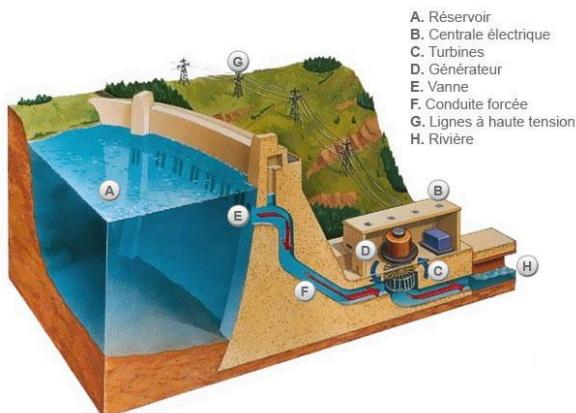
VIII-A - Petit rappel sur l'électricité

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* (c.-à-d. 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 se note *P* et se mesure en *Watts* (W).

NDLR. Sur un barrage, plus la hauteur de chute est importante et plus le débit de l'eau est élevé, et plus la puissance hydraulique transmise à la turbine pour la faire tourner sera grande.

De façon analogue, la puissance électrique *P* est le produit de la tension *U* et de l'intensité *I*.

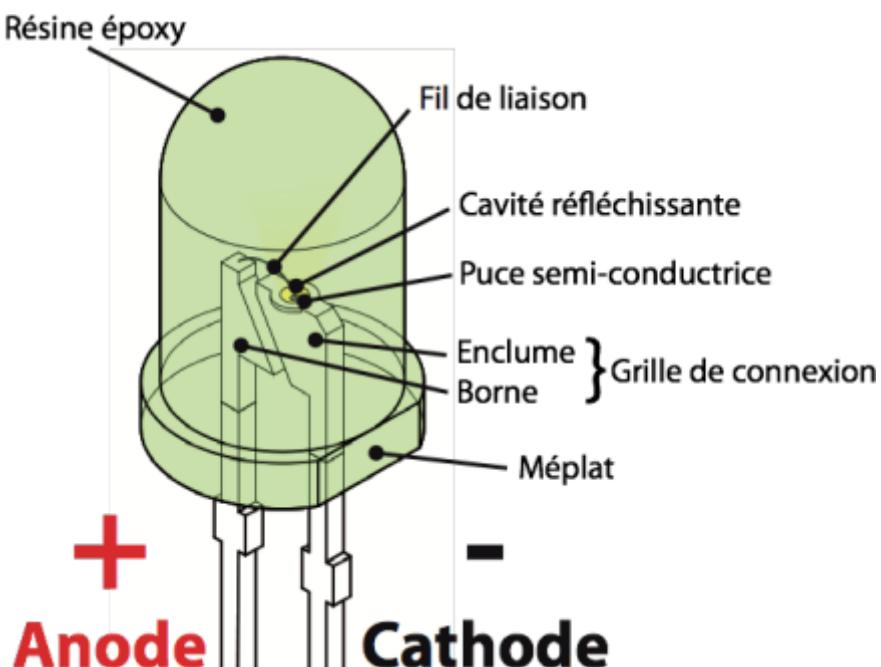
$$P = U \cdot I$$

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=mItO3I82lc0> ;
- Site pédagogique sur l'électricité Hydro-Québec : <http://www.hydroquebec.com/comprendre/> ;
- La bataille de l'électricité : <https://www.youtube.com/watch?v=rbVqoJu6bQ8>.

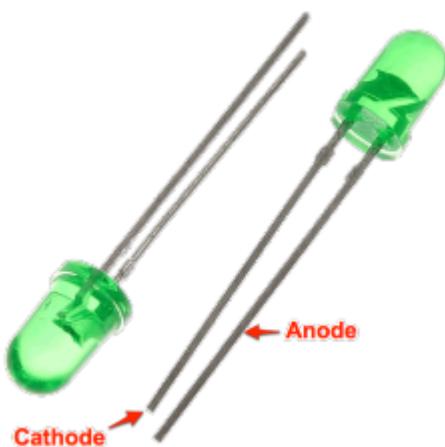
VIII-B - Les diodes

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 griller la LED.

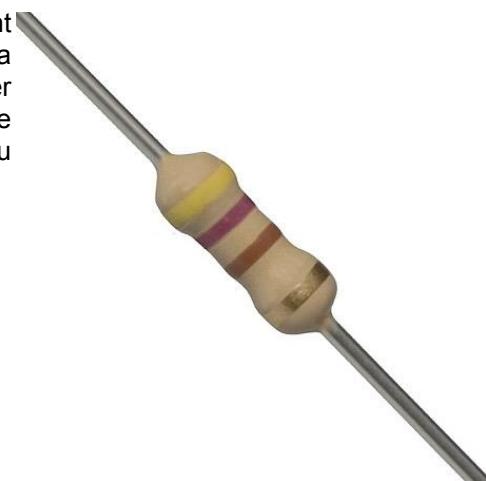


En utilisant divers matériaux semi-conducteurs, on fait varier la couleur de la lumière émise par la LED. Il existe enfin une grande variété de formes de LED.

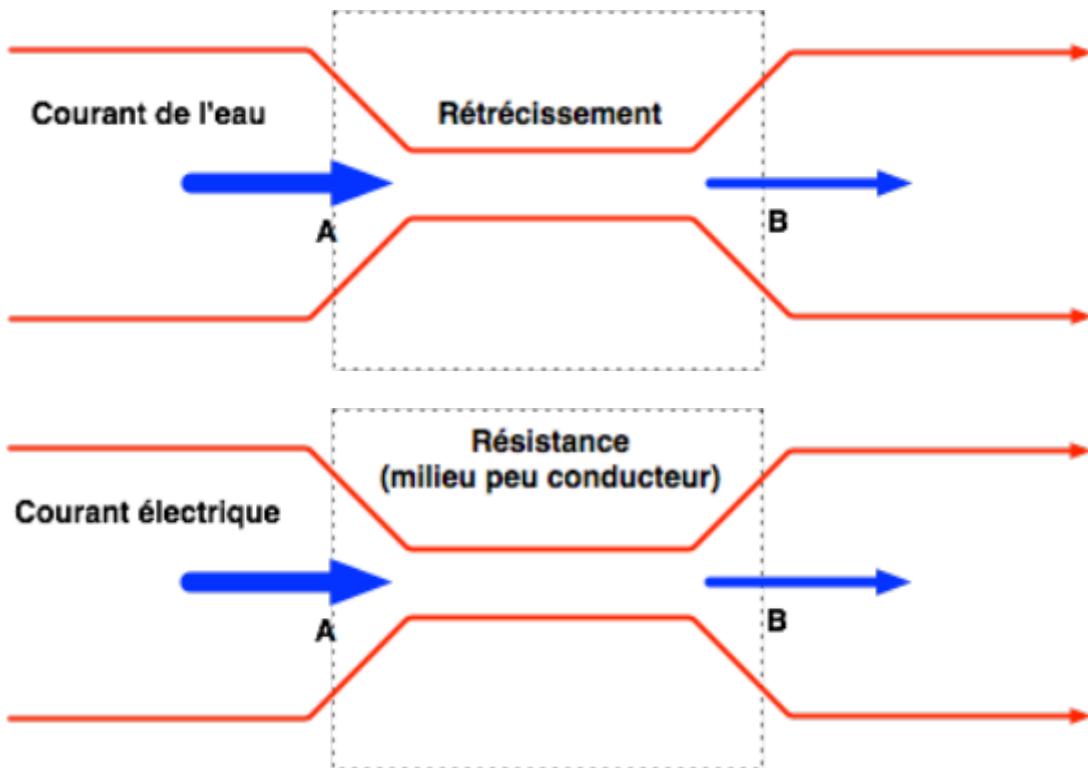


VIII-C - Les résistances

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

$$R = \frac{U}{I}$$

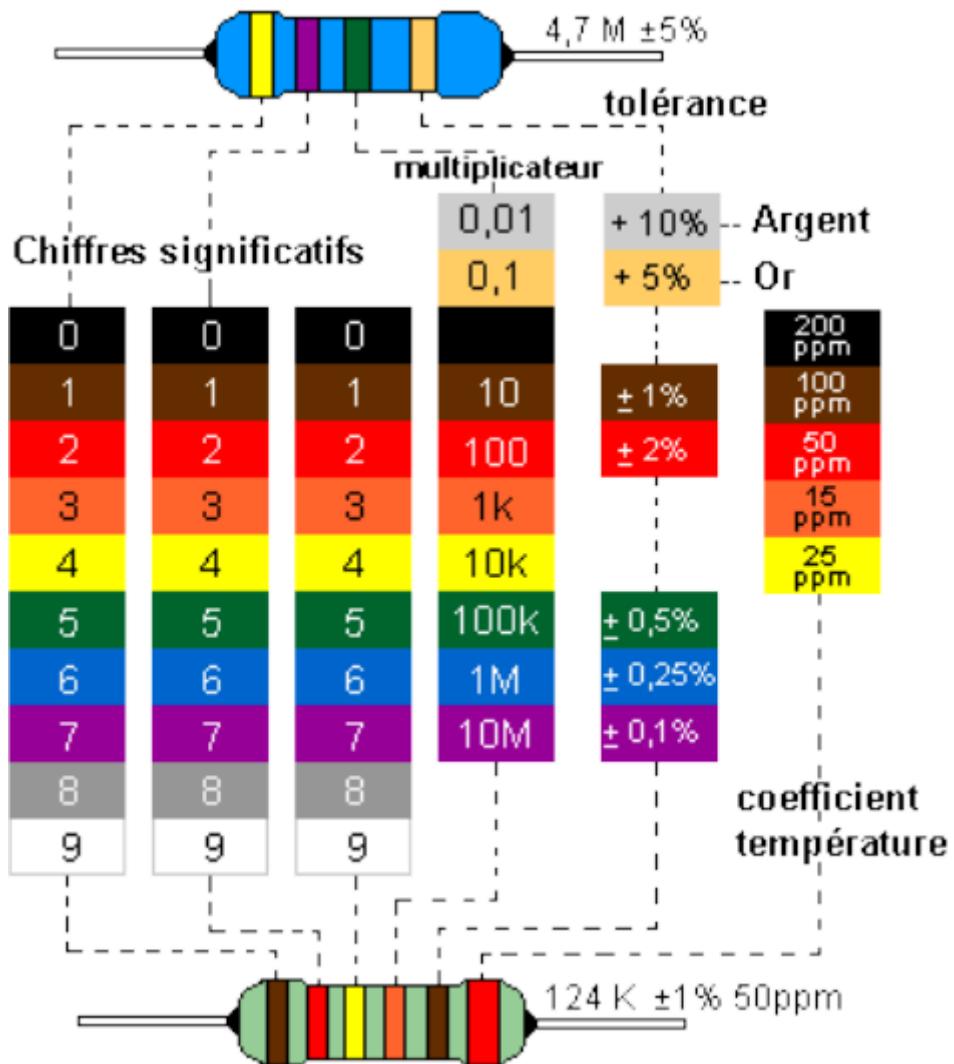
et

$$I = \frac{U}{R}$$

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.



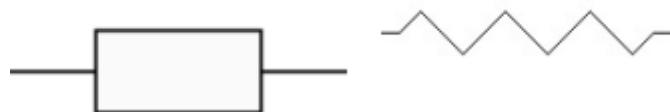
La valeur de la résistance se mesure en Ohms (Ω). La valeur d'une résistance est déterminée par ses bandes de couleurs.



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/passesecret/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/>

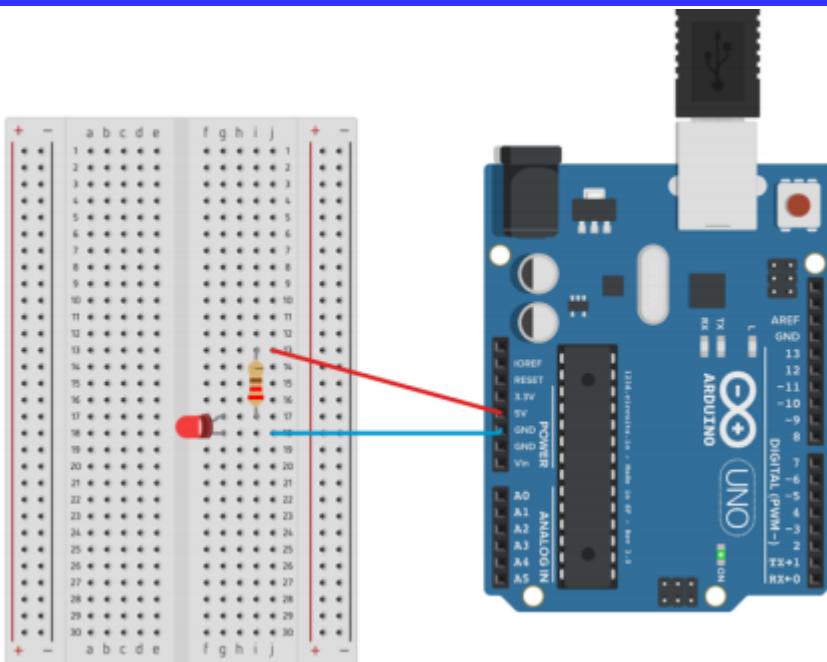


IX - Projet 1 : le circuit électrique

Réalise le circuit suivant :

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

Circuit 1

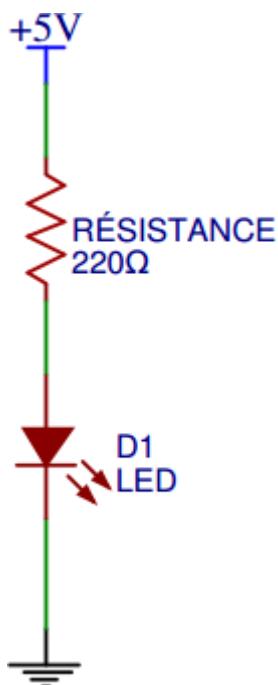


Liste des composants

- 1 LED ;
- 1 résistance de 220 à 470 Ω ;
- 2 câbles.

Observations

Voici ce que cela donne au niveau du schéma électronique.

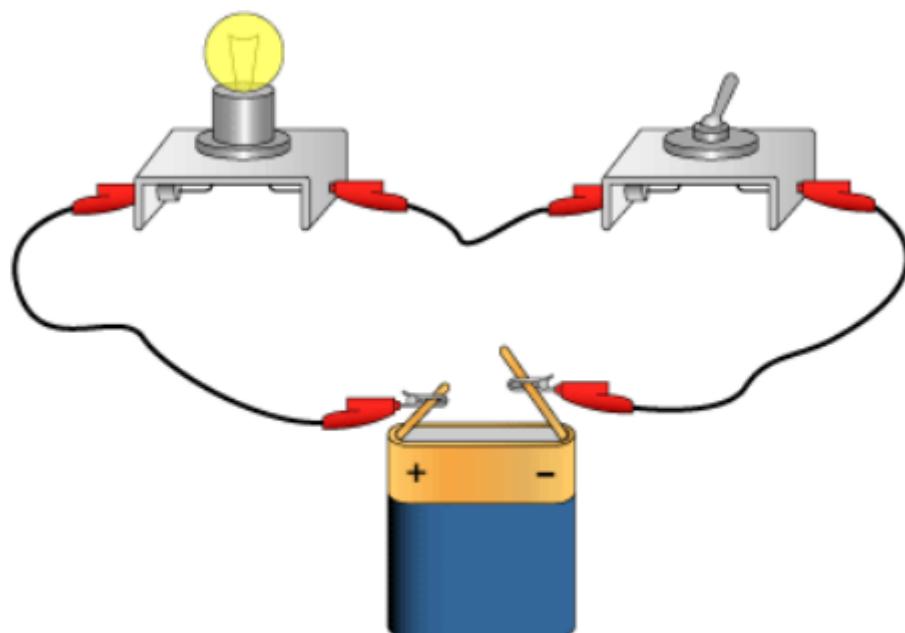


Lorsqu'on branche l'Arduino à l'ordinateur, via le câble USB, il y a 50 % de chances que la LED s'allume. En effet, si elle ne s'allume pas, il faut tourner la LED dans l'autre sens. Sa particularité est que l'électricité ne peut la traverser que dans un sens.

IX-A - Le circuit électrique

Lors de l'exercice 1, tu as réalisé un circuit électrique : tu as relié une source d'électrons 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.

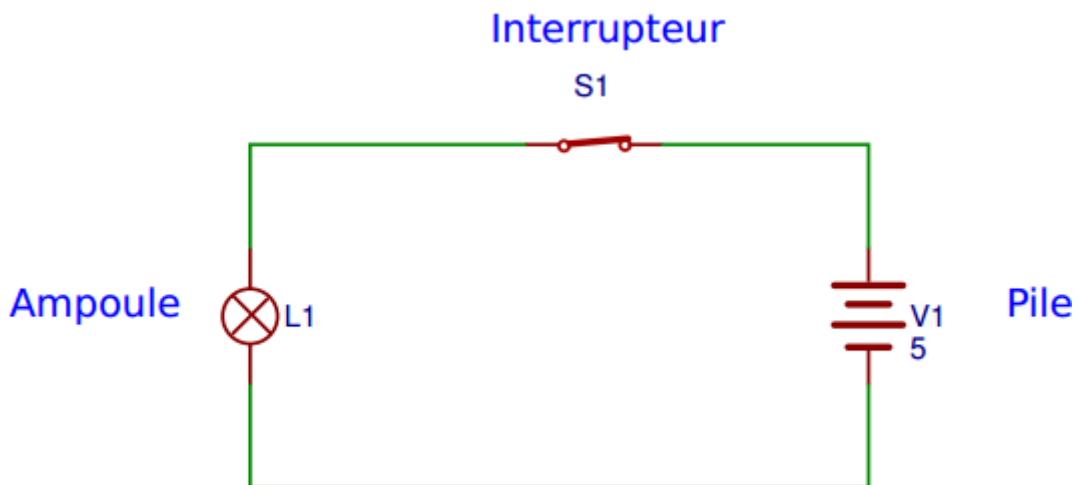
Une pile est constituée d'un milieu contenant de nombreux électrons en trop, et d'un second milieu en manque d'électrons. Quand on relie les deux pôles de la pile (le + et le -) avec un fil électrique (le conducteur), les électrons vont alors se déplacer du milieu riche en électrons vers le milieu pauvre.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



Circuit électrique

Circuit électrique

Voici le schéma électrique du circuit ci-dessus :



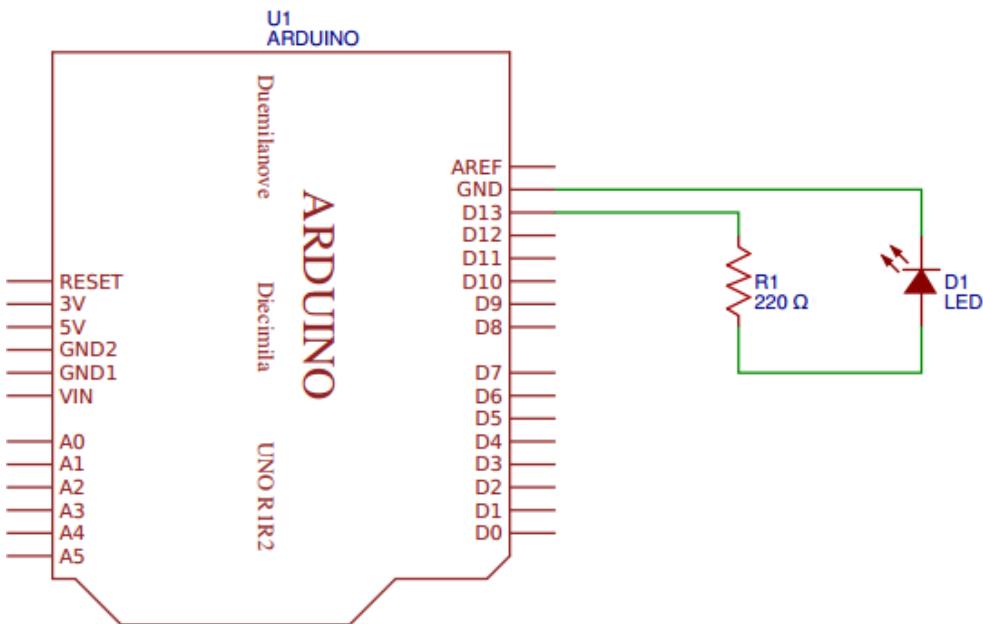
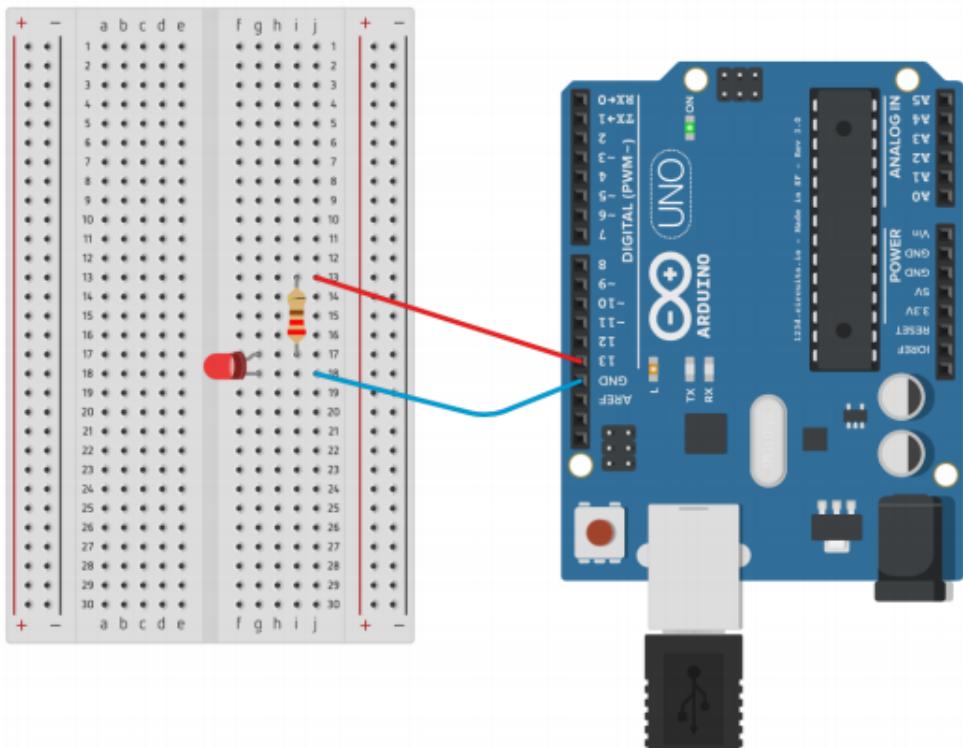
Lorsque l'interrupteur est enclenché, on dit que le circuit est **fermé**. Les électrons vont alors se déplacer et l'énergie de ce déplacement pourra être exploitée pour allumer une lampe ou faire fonctionner un moteur, par exemple.

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.

X - 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

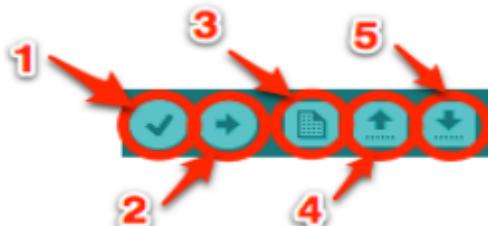
- 1 LED ;
- 1 résistance de 220 à 470 Ω ;
- 2 câbles.

X-A - Le logiciel Arduino IDE

La programmation se fait dans le logiciel Arduino IDE :



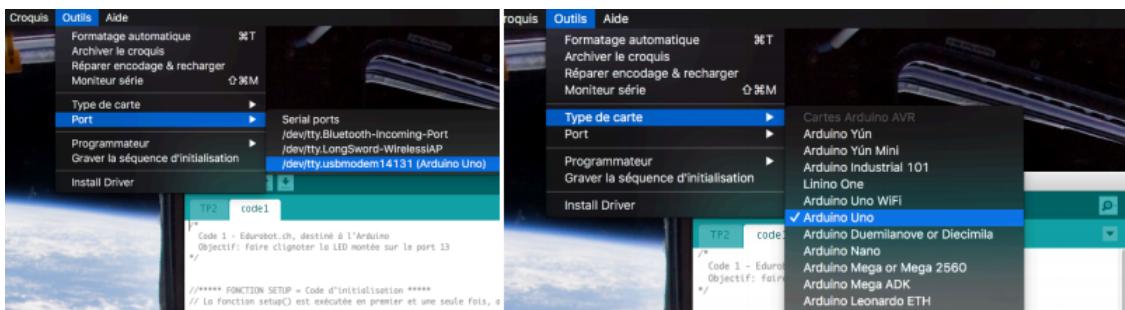
Le menu



- 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_usbmodemXXXXXX`) 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

```

1. /*
2.   Code 1 - Edurobot.ch, destiné à l'Arduino
3.   Objectif: faire clignoter la LED montée sur la broche 13
4. */
5.
6. //***** FONCTION SETUP = Code d'initialisation *****
7. // La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme
8.
9. void setup()      // début de la fonction setup()
10. {
11.     pinMode(13, OUTPUT); // Initialise la broche 13 comme sortie
12.     Serial.begin(9600); // Ouvre le port série à 9600 bauds
13. }                  // fin de la fonction setup()
14.
15. //***** FONCTION LOOP = Boucle sans fin = cœur du programme *****
16. // la fonction loop() s'exécute sans fin en boucle aussi longtemps que l'Arduino est sous tension
17.
18. void loop()        // début de la fonction loop()
19. {
20.     digitalWrite(13, HIGH); // Met la broche 13 au niveau haut = allume la LED
21.     delay(500);           // Pause de 500 ms
22.     digitalWrite(13, LOW); // Met la broche 13 au niveau bas = éteint la LED
23.     delay(500);           // Pause 500 ms
24. }                  // 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 220 Ω sert à abaisser la tension et éviter que la LED ne grille.

Déboguer

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

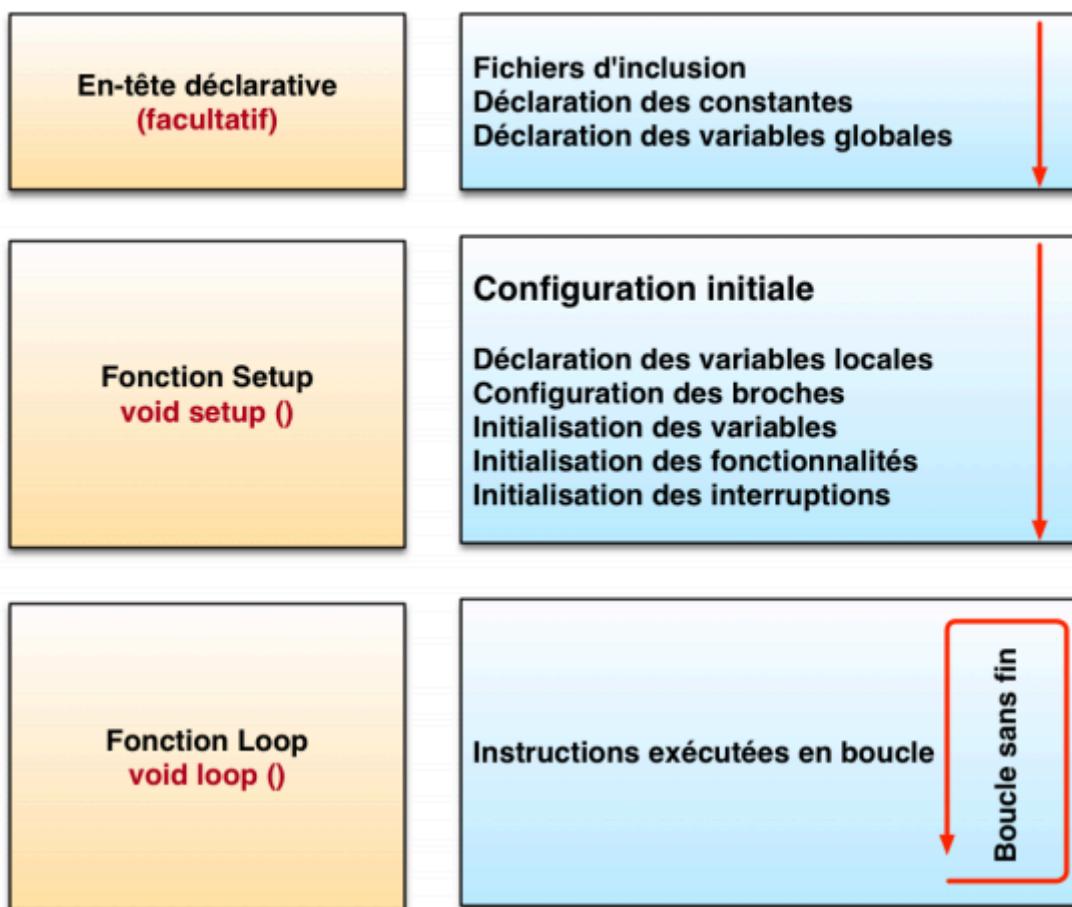
Le déroulement du programme

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.

Déroulement du programme



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 déclaratif, fonction `setup` et fonction `loop`.

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

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.

```

1. void setup()
2. {
3. }
4. void loop()
5. {
6. }
```

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 :

```
1. void setup()
2. {
3. }
```

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 :

```
1. void loop()
2. {
3. }
```

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, elle est exécutée à nouveau, 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 code qui seront ignorées par le programme. Elles ne servent à rien lors de l'exécution du programme. Ils permettent d'annoter et de commenter le programme.

Ligne unique de commentaire :

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

Ligne ou paragraphe sur plusieurs lignes :

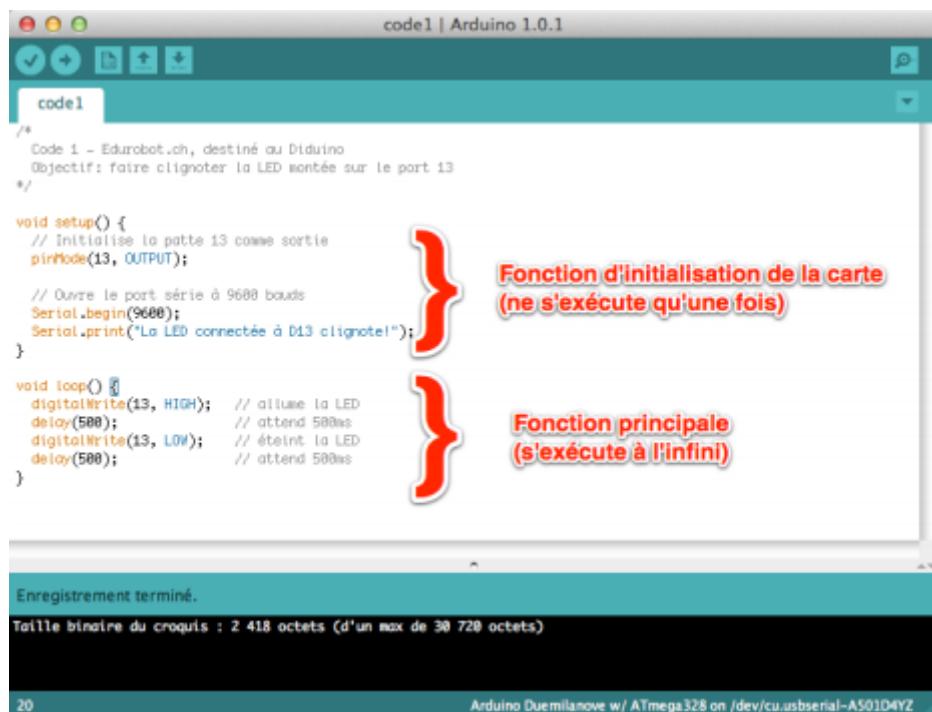
```
1. /*cette ligne est un commentaire, sur PLUSIEURS lignes
2. qui sera ignoré par le programme, mais pas par celui qui lit le code ;) */
```

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.



```
code1 | Arduino 1.0.1
code1
/*
  Code 1 - Edurobot.ch, destiné au Diduino
  Objectif: faire clignoter la LED montée sur le port 13
*/
void setup() {
  // Initialise la patte 13 comme sortie
  pinMode(13, OUTPUT);

  // Ouvre le port série à 9600 bauds
  Serial.begin(9600);
  Serial.print("La LED connectée à D13 clignote!");
}

void loop() {
  digitalWrite(13, HIGH); // allume la LED
  delay(500); // attend 500ms
  digitalWrite(13, LOW); // éteint la LED
  delay(500); // attend 500ms
}

Enregistrement terminé.
Taille binaire du croquis : 2 418 octets (d'un max de 30 728 octets)
20
Arduino Due (Milan) w/ ATmega328 on /dev/cu.usbserial-A501D4YZ
```

- 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 **+5 V** 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 $\frac{1}{2}$ 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 500 ms soit $\frac{1}{2}$ seconde.
- Le résultat est donc que la LED s'allume pendant $\frac{1}{2}$ seconde, puis s'éteint pendant une $\frac{1}{2}$ seconde puis s'allume pendant $\frac{1}{2}$ 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 d'interchanger une ligne de code pour dire qu'il faut utiliser une broche en entrée (NDLR. Broche en « lecture ») ou en sortie (NDLR. Broche en « écriture »).

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

```

1. void setup()
2. {
3.     pinMode(13, OUTPUT);
4.     Serial.begin(9600);
5. }
```

Modifions le code

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

Essai 1 :

```

1. digitalWrite(13, HIGH);
2. delay(50);
3. digitalWrite(13, LOW);
4. delay(50);
```

Essai 2 :

```

1. digitalWrite(13, HIGH);
2. delay(500);
3. digitalWrite(13, LOW);
4. delay(2000);
```

Essai 3 :

```

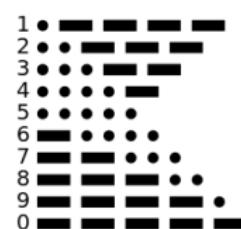
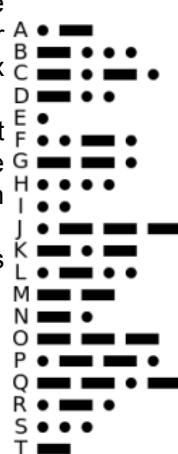
1. digitalWrite(13, HIGH);
2. delay(50);
3. digitalWrite(13, LOW);
4. delay(50);
5. digitalWrite(13, HIGH);
6. delay(1000);
7. digitalWrite(13, LOW);
8. delay(1000);
```

X-A-1 - Exercice : lancer un SOS



Code Morse international

1. Un tiret est égal à trois points.
2. L'espacement entre deux éléments d'une même lettre est égal à un point.
3. L'espacement entre deux lettres est égal à trois points.
4. L'espacement entre deux mots est égal à sept points.



Code de l'exercice SOS

```

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

Code de l'exercice SOS

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

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

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

    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 200 ms

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

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

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

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

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

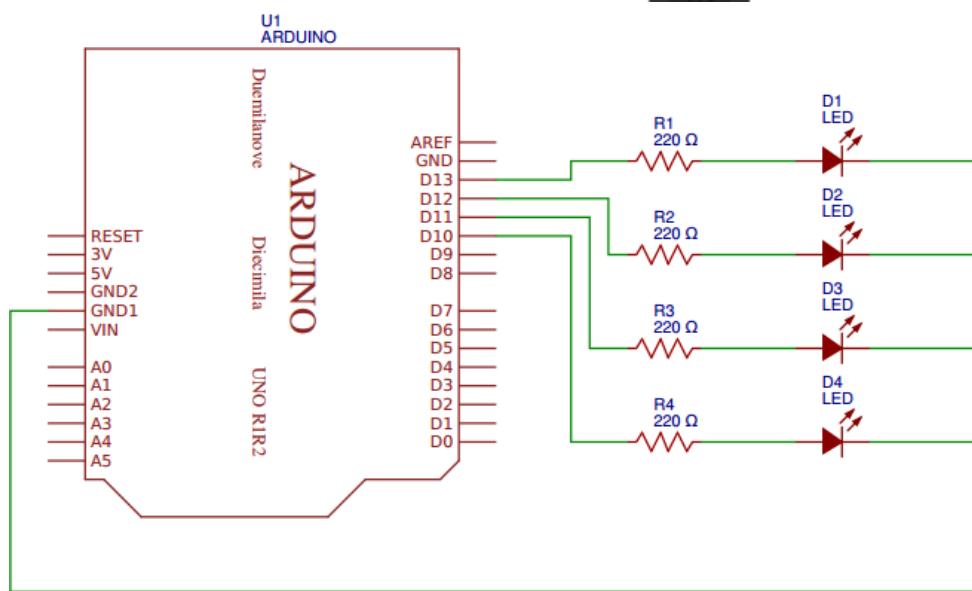
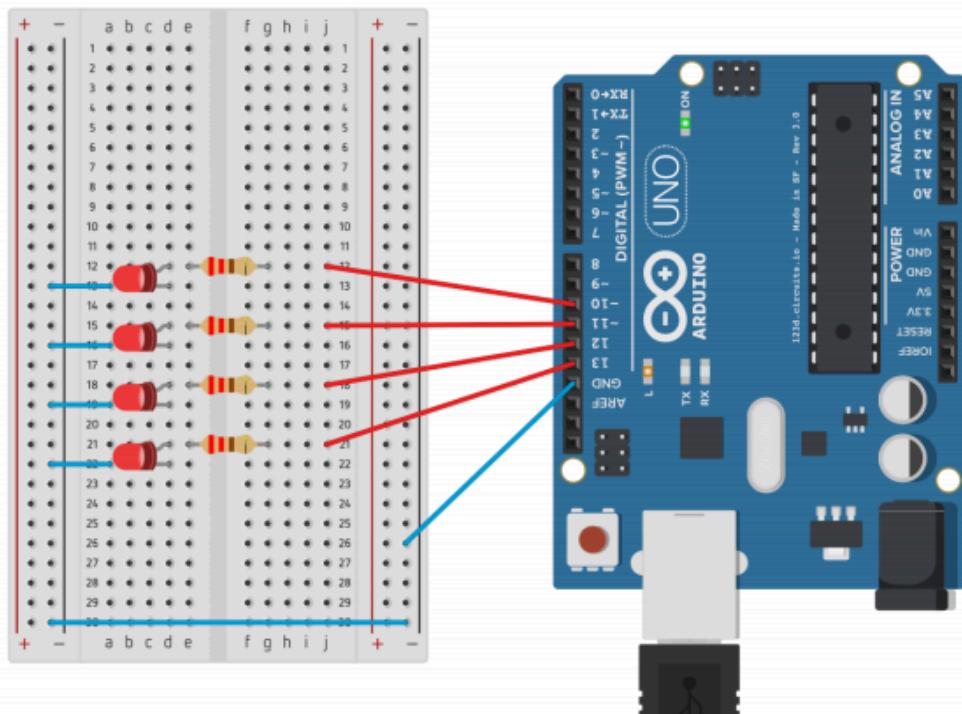
} // fin de la fonction loop()

```

XI - Projet 3 : faire clignoter quatre LED

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

Circuit 3



Liste des composants

- 4 LED ;
- 4 résistances de 220 à 470 Ω ;
- 6 câbles.

Ce code fait clignoter les 4 LED en même temps.

Code 2 : faire clignoter 4 LED

```

1. /*
2.   Code 2 - Edurobot.ch, destiné à l'Arduino
3.   Objectif: faire clignoter les 4 LED montées sur les broches 10 à 13
4. */
5.
6. //***** FONCTION SETUP = Code d'initialisation *****

```

Code 2 : faire clignoter 4 LED

```

7. // La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme
8.
9. void setup()      // début de la fonction setup()
10. {
11.     pinMode(10, OUTPUT);    // Initialise la broche 10 comme sortie
12.     pinMode(11, OUTPUT);    // Initialise la broche 11 comme sortie
13.     pinMode(12, OUTPUT);    // Initialise la broche 12 comme sortie
14.     pinMode(13, OUTPUT);    // Initialise la broche 13 comme sortie
15.
16.     Serial.begin(9600);    // Ouvre le port série à 9600 bauds
17. }                  // fin de la fonction setup()
18.
19. //***** FONCTION LOOP = Boucle sans fin = cœur du programme *****
20. // la fonction loop() s'exécute sans fin en boucle aussi longtemps que l'Arduino est sous tension
21.
22. void loop()      // début de la fonction loop()
23. {
24.     digitalWrite(10, HIGH); // Met la broche 10 au niveau haut = allume la LED
25.     digitalWrite(11, HIGH); // Met la broche 11 au niveau haut = allume la LED
26.     digitalWrite(12, HIGH); // Met la broche 12 au niveau haut = allume la LED
27.     digitalWrite(13, HIGH); // Met la broche 13 au niveau haut = allume la LED
28.     delay(500);           // Pause de 500 ms
29.     digitalWrite(10, LOW); // Met la broche 10 au niveau bas = éteint la LED
30.     digitalWrite(11, LOW); // Met la broche 11 au niveau bas = éteint la LED
31.     digitalWrite(12, LOW); // Met la broche 12 au niveau bas = éteint la LED
32.     digitalWrite(13, LOW); // Met la broche 13 au niveau bas = éteint la LED
33.     delay(500);           // Pause 500 ms
34. }                  // fin de la fonction loop()

```

Code 3 : réaliser un chenillard à 4 LED.

```

1. /*
2.     Code 3 - Edurobot.ch, destiné à l'Arduino
3.     Objectif : faire un chenillard à 4 LED montées sur les broches 10 à 13
4. */
5.
6. void setup() // début de la fonction setup()
7. {
8.     pinMode(10, OUTPUT);    // Initialise la broche 10 comme sortie
9.     pinMode(11, OUTPUT);    // Initialise la broche 11 comme sortie
10.    pinMode(12, OUTPUT);    // Initialise la broche 12 comme sortie
11.    pinMode(13, OUTPUT);    // Initialise la broche 13 comme sortie
12.
13.    Serial.begin(9600);    // Ouvre le port série à 9600 bauds
14. }                  // fin de la fonction setup()
15.
16. void loop() // début de la fonction loop()
17. {
18.     digitalWrite(10, HIGH); // Met la broche 10 au niveau haut = allume la LED
19.     digitalWrite(11, LOW); // Met la broche 11 au niveau bas = éteint la LED
20.     digitalWrite(12, LOW); // Met la broche 12 au niveau bas = éteint la LED
21.     digitalWrite(13, LOW); // Met la broche 13 au niveau bas = éteint la LED
22.
23.     delay(100);           // Pause de 100 ms
24.
25.     digitalWrite(10, LOW); // Met la broche 10 au niveau bas = éteint la LED
26.     digitalWrite(11, HIGH); // Met la broche 11 au niveau haut = allume la LED
27.
28.     delay(100);           // Pause de 100 ms
29.
30.     digitalWrite(11, LOW); // Met la broche 11 au niveau bas = éteint la LED
31.     digitalWrite(12, HIGH); // Met la broche 12 au niveau haut = allume la LED
32.
33.     delay(100);           // Pause de 100 ms
34.
35.     digitalWrite(12, LOW); // Met la broche 12 au niveau bas = éteint la LED
36.     digitalWrite(13, HIGH); // Met la broche 13 au niveau haut = allume la LED
37.
38.     delay(100);           // Pause de 100 ms

```

Code 3 : réaliser un chenillard à 4 LED.

```

39.
40.     digitalWrite(13, LOW); // Met la broche 13 au niveau bas = éteint la LED
41.     digitalWrite(12, HIGH); // Met la broche 12 au niveau haut = allume la LED
42.
43.     delay(100); // Pause de 100 ms
44.
45.     digitalWrite(12, LOW); // Met la broche 12 au niveau bas = éteint la LED
46.     digitalWrite(11, HIGH); // Met la broche 11 au niveau haut = allume la LED
47.
48.     delay(100); // Pause de 100ms
49. }           // fin de la fonction loop

```

XII - Projet 4 : introduction aux variables

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

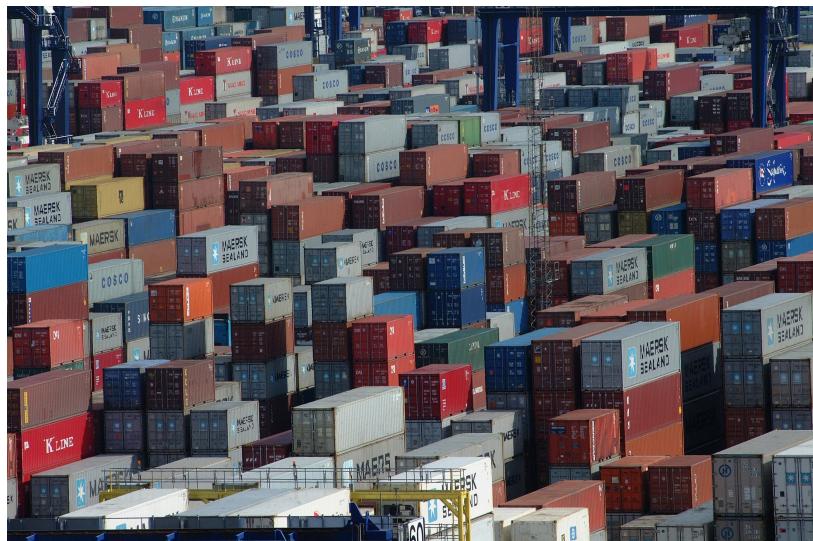
Imaginons un nombre dont nous devons nous souvenir. Ce nombre est stocké dans un espace de stockage de la mémoire vive (RAM) du microcontrôleur. **Chaque espace de stockage est identifié de manière unique.**

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 3 GHz multicœur, 8 Go de RAM ! Ici, il s'agit d'un système qui fonctionne avec un CPU à 16 MHz (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 n'y en a plus, y en a plus ! Dans un même volume, on peut stocker bien plus de petits cartons que de 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 x 10 ³⁸ à +3.4 x 10 ³⁸	32 bits	4 octets
double	décimale	-3.4 x 10 ³⁸ à +3.4 x 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.

```
1. char x = 0;
```

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

```
1. 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 int	entier non négatif	0 à 65 535	16 bits	2 octets
unsigned long	entier non négatif	0 à 4 294 967 295	32 bits	4 octets

Une des particularités du langage Arduino est qu'il accepte un nombre plus important de types de variables, listées dans ce tableau :

Type variable	Type nombre stocké	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
byte	entier négatif	0 à 255	8 bits	1 octet
word	entier négatif	0 à 65 535	16 bits	2 octets
boolean	entier négatif	0 à 1	1 bit	1 octet

XII-B - Définir les broches du microcontrôleur

Jusqu'à maintenant, nous avons identifié les broches du microcontrôleur à l'aide de leur numéro, comme dans l'exemple suivant : `pinMode(13, OUTPUT);`. Cela ne pose pas de problème quand on a une ou deux LED 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.

```
1. 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 -2 147 483 648 à +2 147 483 647, 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 -2 147 483 648 à +2 147 483 647. 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	Broche définie
<pre>1. void setup() { 2. pinMode(13, OUTPUT); 3. // Initialise la broche 4. // 13 comme sortie 5. Serial.begin(9600); 6. 7. void loop() { 8. digitalWrite(13, HIGH); 9. delay(500); 10. digitalWrite(13, LOW); 11. delay(500); 12. }</pre>	<pre>1. const int led1= 13; 2. //La broche 13 devient led1 3. 4. void setup() { 5. pinMode(led1, OUTPUT); 6. // Initialise led1 comme 7. // broche de sortie 8. Serial.begin(9600); 9. 10. void loop() { 11. 12. digitalWrite(led1, HIGH); 13. delay(500); 14. digitalWrite(led1, LOW); 15. }</pre>

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.

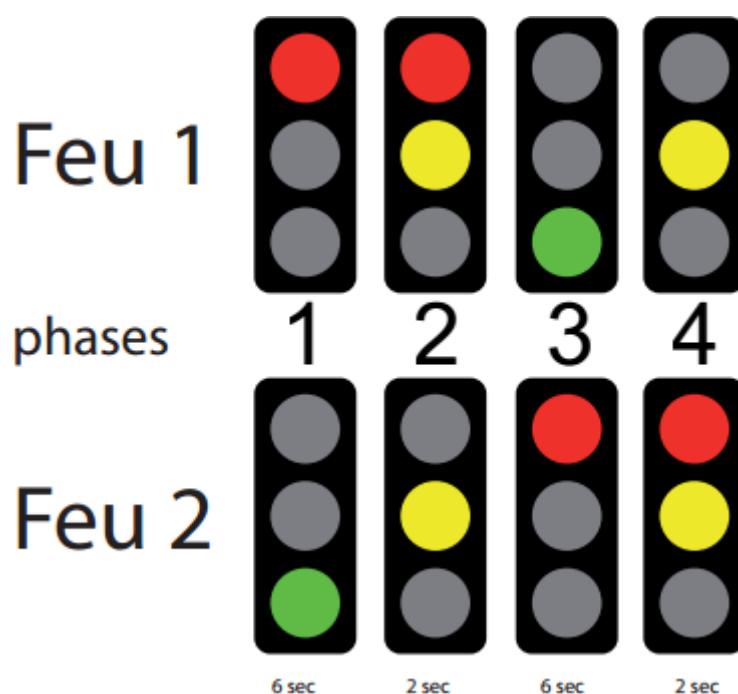
Comme pour les variables, nous pouvons donner n'importe quel nom aux broches.

XIII - 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.

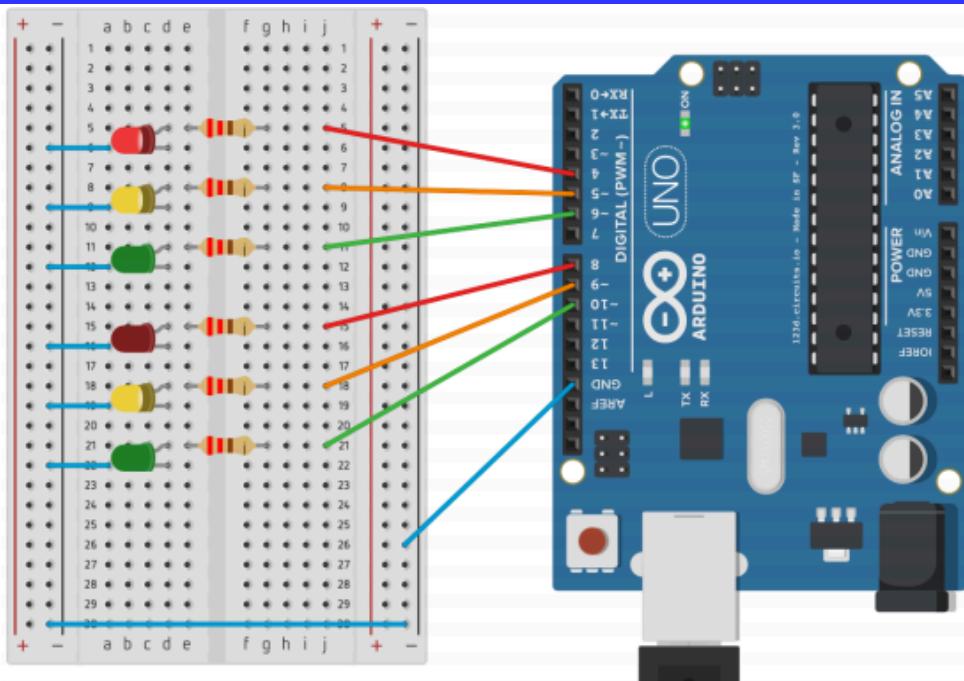
L'objectif de cet exercice est de créer deux feux de circulation et de les faire fonctionner de manière synchrone.

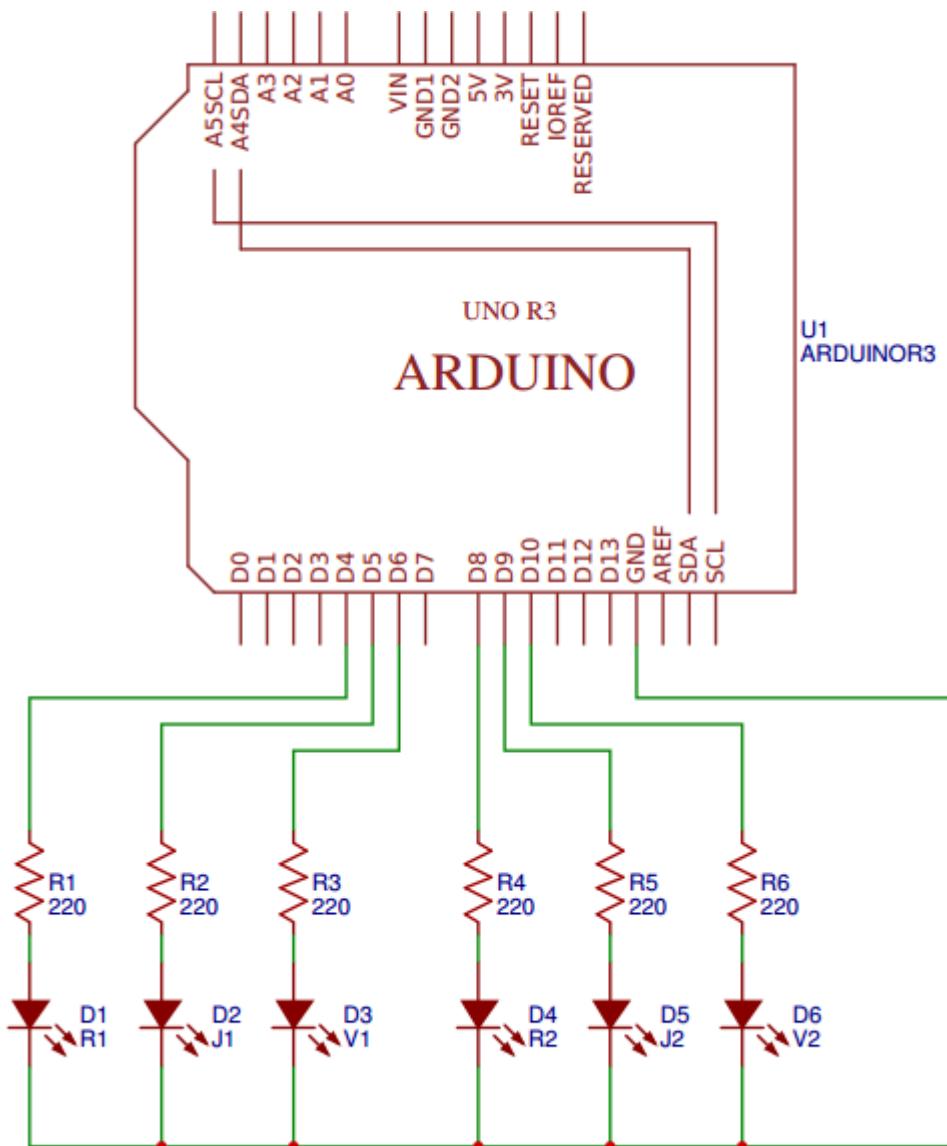
Voici les phases de deux feux de circulation que tu dois recréer :



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



Circuit 4




Liste des composants

- 2 LED rouges ;
- 2 LED jaunes ou orange ;
- 2 LED vertes ;
- 6 résistances de 220 à 470 Ω .

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

```

1. /*
2.  Code 4 - Edurobot.ch, destiné à l'Arduino
3.  Objectif : gérer des feux de circulation
4. */
5.
6. // On définit les variables pour chaque broche
7. //FEU 1
8. const int R1 = 4;      // La broche 4 devient le feu rouge 1
9. const int J1 = 5;      // La broche 3 devient le feu jaune 1
10. const int V1 = 6;     // La broche 2 devient le feu vert 1
11. //FEU2
12. const int R2 = 8;      // La broche 8 devient le feu rouge 2
13. const int J2 = 9;      // La broche 9 devient le feu jaune 2
14. const int V2 = 10;     // La broche 10 devient le feu vert 2
15. //TEMPS
16. int timer1 = 2000;    // Le temps est fixé à 2 secondes
17. int timer2 = 6000;    // Le temps est fixé à 6 secondes
18.
19. void setup() {
20.     // On initialise les sorties
21.     pinMode(R1, OUTPUT);
22.     pinMode(J1, OUTPUT);
23.     pinMode(V1, OUTPUT);
24.
25.     pinMode(R2, OUTPUT);
26.     pinMode(J2, OUTPUT);
27.     pinMode(V2, OUTPUT);
28. }
29.
30. void loop() {
31.     // Phase 1 : R1 et V2 sont allumés, R2, J1 et J2 sont éteints
32.     digitalWrite(R2, LOW);    // R2 éteint
33.     digitalWrite(J1, LOW);   // J1 éteint
34.     digitalWrite(J2, LOW);   // J2 éteint
35.     digitalWrite(R1, HIGH);  // R1 allumé
36.     digitalWrite(V2, HIGH);  // V2 allumé
37.     delay(timer2);         // durée: 6000 millisecondes, soit 6 secondes
38.
39.     // Phase 2 : R1, J1, J2 allumés et V2 éteint
40.     digitalWrite(V2, LOW);   // V2 éteint
41.     digitalWrite(J1, HIGH);  // J1 allumé
42.     digitalWrite(J2, HIGH);  // J2 allumé
43.     delay(timer1);         // durée: 2000 millisecondes, soit 2 secondes
44.
45.     // Phase 3 : R1, J1, J2 éteints et V1 et R2 allumés
46.     digitalWrite(R1, LOW);  // R1 éteint
47.     digitalWrite(J1, LOW);  // J1 éteint
48.     digitalWrite(J2, LOW);  // J2 éteint
49.     digitalWrite(V1, HIGH); // V1 allumé
50.     digitalWrite(R2, HIGH); // R2 allumé
51.     delay(timer2);
52.
53.     // Phase 4 : V1 éteint et J1, J2 et R2 allumés
54.     digitalWrite(V1, LOW);  // V1 éteint
55.     digitalWrite(J1, HIGH); // J1 allumé
56.     digitalWrite(J2, HIGH); // J2 allumé
57.     digitalWrite(R2, HIGH); // R2 allumé
58.     delay(timer1);
59. }
```

XIII-A - 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.

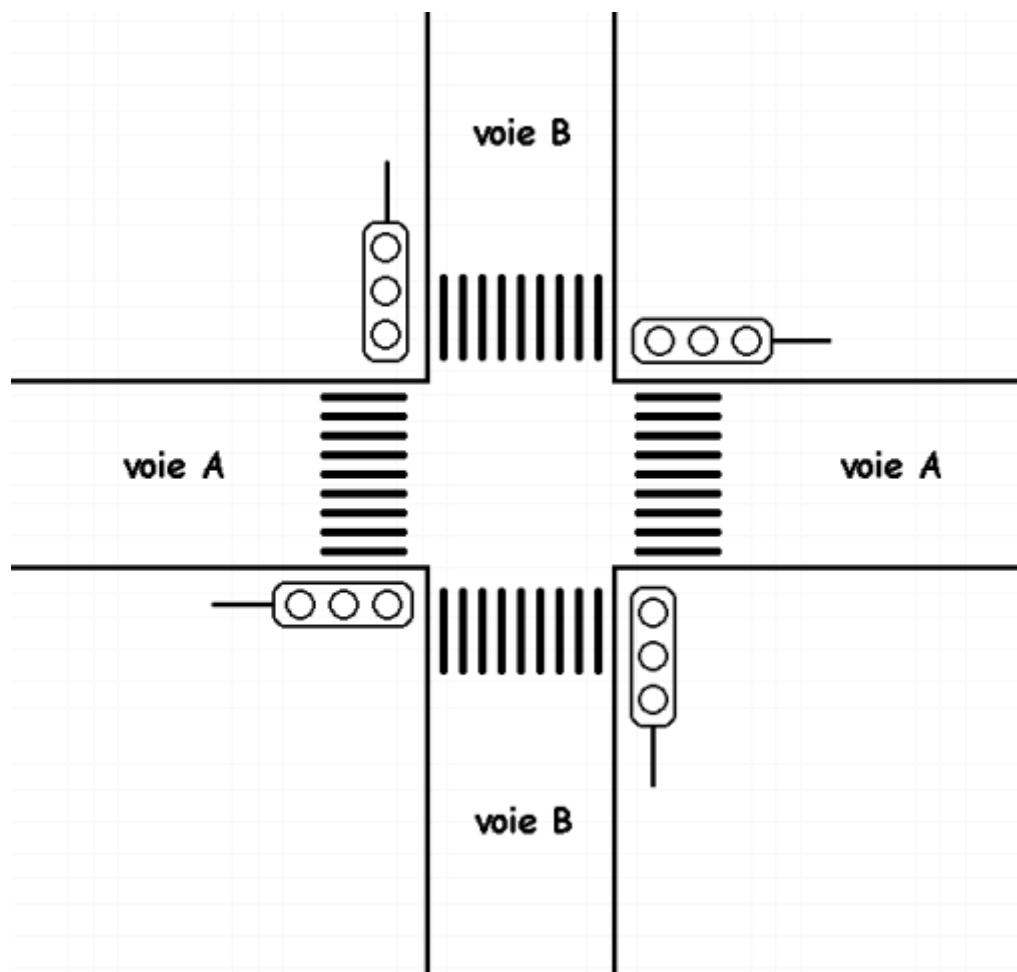
Variante 2

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 :



XIV - Projet 6 : l'incrémentation

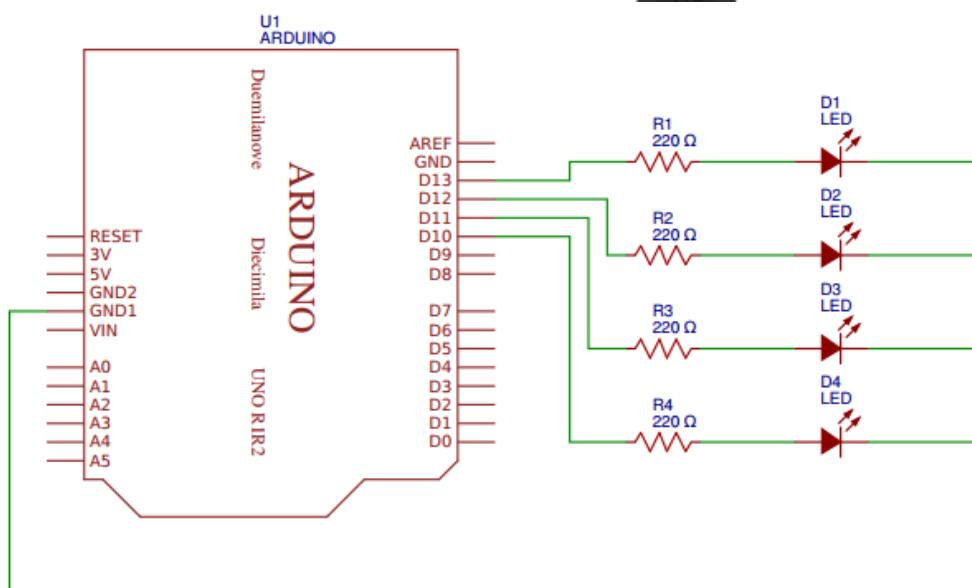
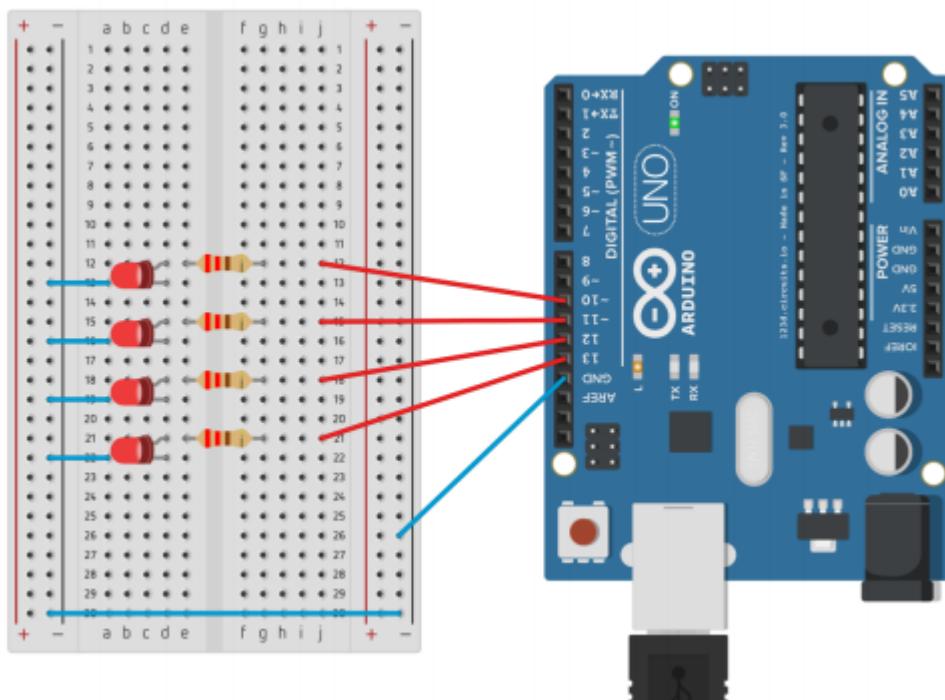
Il est possible d'appliquer aux valeurs des variables diverses opérations mathématiques. Commençons tout de suite par un petit exemple : l'*incrémentation*. Il s'agit simplement d'ajouter 1 à la variable. À chaque fois qu'on répète le code, on prend la valeur de la variable et on y ajoute 1.

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 LED rouges ;
- 4 résistances de 220 à 470 Ω ;

Code 5 : faire clignoter 10 fois la LED 13

```

1. /*
2.   Code 5 - Edurobot.ch, destiné à l' Arduino
3.   Objectif : faire clignoter 10 fois la LED montée sur le port 13
4. */
5.

```

Code 5 : faire clignoter 10 fois la LED 13

```

6. //***** EN-TÈTE DÉCLARATIF *****
7. // On déclare les variables, les constantes...
8.
9. byte compteur; //On définit la variable "compteur"
10. const int led1= 13; //On renomme la broche 10 en "led1"
11.
12. //***** FONCTION SETUP = Code d'initialisation *****
13. // La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme
14.
15. void setup()
16. {
17.     pinMode(led1, OUTPUT);           // Initialise la broche 13 comme sortie
18.     Serial.begin(9600);           // Ouvre le port série à 9600 bauds
19.     // Exécute le programme entre accolades en partant de zéro et en incrémentant à chaque
    fois la
20.     // valeur de +1: 0+1/2+1/3+1... jusqu'à ce que la variable « compteur » soit égale à 9
    (plus petit que 10).
21.     for(compteur=0 ; compteur<10 ; compteur++)
22.     { // début du programme exécuté 10 fois
23.         digitalWrite(led1, HIGH); // allume la LED
24.         delay(500);             // attend 500ms
25.         digitalWrite(led1, LOW); // éteint la LED
26.         delay(500);             // attend 500ms
27.     }
28.     // fin du programme exécuté 10 fois
29. }
30.
31. void loop() {
32. } // 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 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 :

```
21. for(compteur=0 ; compteur<10 ; compteur++)
```

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 comprise 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. À chaque fois, on ajoute +1 à ta variable *compteur*, jusqu'à ce qu'on arrive à 9. À 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 le code suivant qui est exécuté 10 fois :

```

23.     digitalWrite(Led1, HIGH); // allume la LED
24.     delay(500);           // attend 500 ms
25.     digitalWrite(Led1, LOW); // éteint la LED
26.     delay(500);           // attend 500 ms

```

Voici les parties d'une déclaration **for** :

Déclaration de la variable (optionnel)



L'initialisation est effectuée en premier et une seule fois. À 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.

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

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

```

1. /*
2.   Code 6 - Edurobot.ch, destiné à l'Arduino
3.   Objectif : faire un chenillard à 4 LED montées sur les ports 10 à 13
4. */
5.
6. //***** EN-TÊTE DÉCLARATIF *****
7.
8. // Définition de la variable « temps »
9. int timer = 100; // Durée, en millisecondes
10.
11. //***** FONCTION SETUP = Code d'initialisation *****
12. // La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme
13.
14. void setup() {
15.     // Déclaration des broches 10 à 13 à l'aide d'un for et d'un incrément.
16.     for (int thisPin = 10; thisPin < 14; thisPin++) {
17.         pinMode(thisPin, OUTPUT);
18.     }
19. }
20.
21. void loop() {
22.     // Boucle de la broche 10 à la broche 13 :
23.     for (int thisPin = 10; thisPin < 14; thisPin++) { // Incrément faisant passer la variable
24.         thisPin de 10 à 13
25.         digitalWrite(thisPin, HIGH); // Allumer la LED
26.         delay(timer);           // Durée
27.         digitalWrite(thisPin, LOW); // Éteindre la LED
28.     }
29.     // Boucle de la broche 13 à la broche 10 :
30.     for (int thisPin = 13; thisPin >= 10; thisPin--) { // Décrément faisant passer la
31.     variable thisPin de 13 à 10
32.         digitalWrite(thisPin, HIGH); // Allumer la LED
33.         delay(timer);           // Durée
34.         digitalWrite(thisPin, LOW); // Éteindre la LED;
35.     }
36. }

```

Analyse du code

Regardons maintenant un peu ce code, en particulier :

```
16. for (int thisPin = 10; thisPin < 14; thisPin++) {
17.     pinMode(thisPin, OUTPUT);
18. }
```

Sa première particularité est la manière de définir les broches. Au lieu d'accumuler les `pinMode` (xxx, `OUTPUT`);, on crée une variable de type `int` qu'on appelle `thisPin` (7) et donc la valeur initiale est de 10. Un `for` avec un incrément (`++`) permet d'incrémenter la valeur de la variable de 10 à 14. Il suffit ensuite de

remplacer dans le `pinMode` le numéro de la broche 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 :

```
23. for (int thisPin = 10; thisPin < 14; thisPin++) { // Incrémentation faisant passer la variable
   thisPin de 10 à 13
24.     digitalWrite(thisPin, HIGH); // Allumer la LED
25.     delay(timer);           // Durée
26.     digitalWrite(thisPin, LOW); // Éteindre la LED
27. }
```

Comme pour la définition des broches en `OUTPUT`, on utilise la boucle `for` pour incrémenter la variable `thisPin` de 10 à 13. Ainsi, on allume les LED connectées sur les broches 10 à 13 l'une après l'autre.

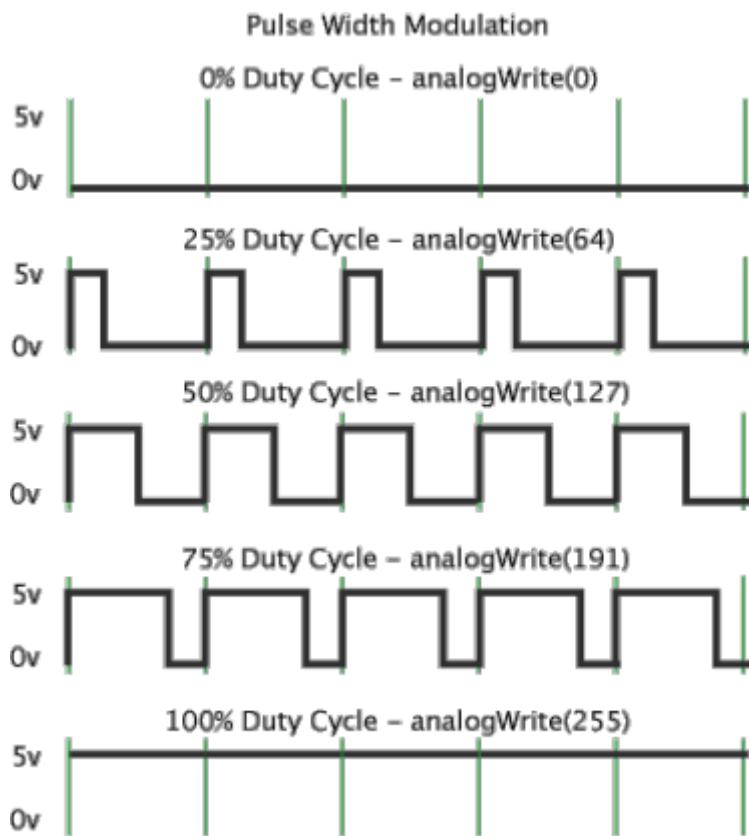
Enfin dans la troisième partie du code, on va allumer les LED de la broche 13 à la broche 10 avec une boucle `for` et une décrémentation. Pour cela, on remplace le `++` par un `--`.

```
29. // Boucle de la broche 13 à la broche 10:
30. for (int thisPin = 13; thisPin >= 10; thisPin--) { // Décrémentation faisant passer la variable
   thisPin de 13 à 10
31.     digitalWrite(thisPin, HIGH); // Allumer la LED
32.     delay(timer);           // Durée
33.     digitalWrite(thisPin, LOW); // Éteindre la LED;
34. }
```

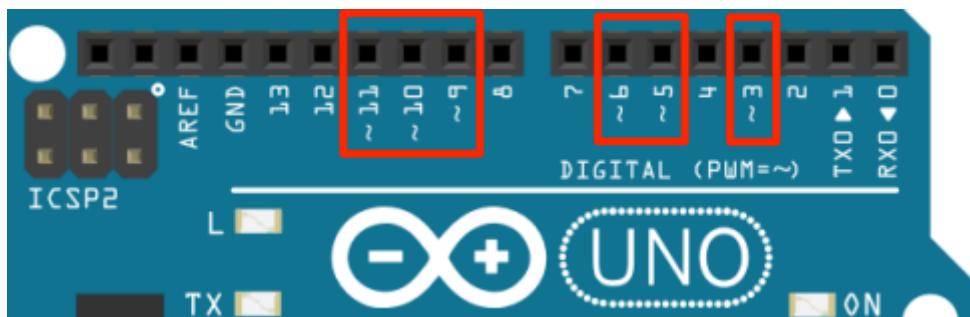
XV - Projet 7 : PWM, variation en douceur d'une LED

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'autres termes, cela signifie qu'on va faire clignoter très vite les LED ; tellement vite que l'œil ne percevra qu'une lumière continue. Mais plus on 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 :

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

$$\text{Valeur en \%} = \frac{\text{Valeur PWM}}{255} \cdot 100$$

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

```

1. /*
2.  Code 7 - Edurobot.ch, destiné à l'Arduino
3.  Objectif : faire varier la luminosité d'une LED sur la broche 10 en modifiant la valeur PWM
4. */
5.
6. //***** EN-TÊTE DÉCLARATIF *****
7.
8. int ledPin = 10; // On renomme la broche 10 en "ledPin"
9. int timer = 100; // On définit une durée de 0, 1 seconde pour la variable timer
10.

```

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

```

11. //***** SETUP *****
12.
13. void setup() {
14.     pinMode(ledPin, OUTPUT);
15. }
16.
17. //***** LOOP *****
18. void loop() {
19.     // LED à 0 %.
20.     analogWrite(ledPin, 0);
21.     delay(timer);
22.
23.     // LED à 19.6 %.
24.     analogWrite(ledPin, 50);
25.     delay(timer);
26.
27.     // LED à 39.2 %.
28.     analogWrite(ledPin, 100);
29.     delay(timer);
30.
31.     // LED à 58.8 %.
32.     analogWrite(ledPin, 150);
33.     delay(timer);
34.
35.     // LED à 78.4 %.
36.     analogWrite(ledPin, 200);
37.     delay(timer);
38.
39.     // LED à 100 %.
40.     analogWrite(ledPin, 255);
41.     delay(timer);
42. }
```

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 boucle `for`, comme nous le verrons dans le code suivant :

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

```

1. /*
2.  * Code 8 - Edurobot.ch, destiné à l'Arduino
3.  * Objectif : faire varier la luminosité d'une LED sur la broche 10
4.  * Adapté de David A. Mellis et Tom Igoe
5. */
6.
7. //***** EN-TÊTE DÉCLARATIF *****
8.
9. int ledPin = 10; //On renomme la broche 10 en "ledPin"
10.
11. //***** SETUP *****
12. void setup() {
13.
14. }
15.
16. //***** LOOP *****
17. void loop() {
18.     // Variation du min au max par addition de 5 jusqu'à 256
19.     for (int fadeValue = 0 ; fadeValue <= 255; fadeValue += 5)
20.     {
21.         // Définition de la valeur de luminosité (de 0 à 255)
22.         analogWrite(ledPin, fadeValue);
23.         // Attente de 30 millisecondes entre chaque palier pour voir l'effet.
24.         delay(30);
25.     }
26. }
```

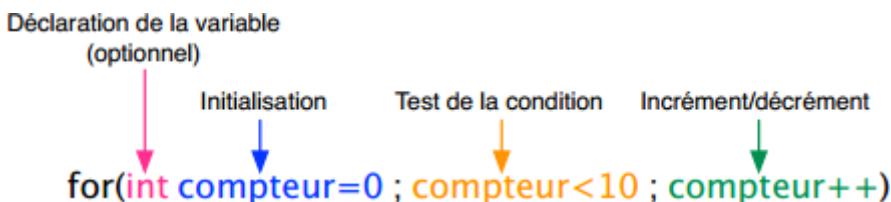
Code 8 : faire varier la luminosité d'une LED en douceur

```

27.     // Variation du max au min par soustraction de 5 depuis 256
28.     for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -= 5)
29.     {
30.         // Définition de la valeur de luminosité (de 0 à 255)
31.         analogWrite(ledPin, fadeValue);
32.         // Attente de 30 millisecondes entre chaque palier pour voir l'effet.
33.         delay(30);
34.     }
35. }
```

Analyse du code

La boucle **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 crée 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 $51 \times 30 = 1530$ 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, grâce 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**.

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 : autre solution pour faire varier la luminosité d'une LED

```

1. /*
2.  * Code 9 - Edurobot.ch, destiné à l'Arduino
3.  * Objectif : faire varier la luminosité d'une LED sur la broche 10
4. */
5.
6. //***** EN-TÊTE DÉCLARATIF *****
7.
8. int ledPin = 10; // On renomme la broche 10 en "ledPin"
9.
10. //***** SETUP *****
11. void setup()
12. {
13. }
14.
15. //***** LOOP *****
16. void loop()
17. {
18.     // Variation du min au max par incrément
19.     for (int fadeValue = 0; fadeValue <= 255; fadeValue++) {
20.         analogWrite(ledPin, fadeValue);
```

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

```

21.     delay(10);
22. }
23.
24. // Variation du max au min par décrément
25. for (int fadeValue = 255; fadeValue >= 0; fadeValue --) {
26.     analogWrite(ledPin, fadeValue);
27.     delay(10);
28. }
29. }
```

XVI - 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 LED.

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`.

XVI-A - 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 +5 V. 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 200 Ω.



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 :



```

1. //Programme vide d'initialisation
2. void setup() {
3. }
4.
5. void loop() {
6. }
```

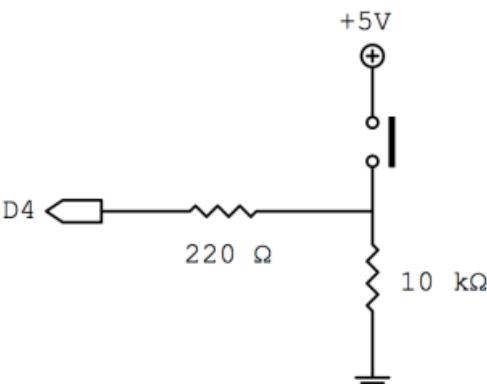
Maintenant que ce point est réglé, voyons comment utiliser le bouton-poussoir avec le microcontrôleur.

XVI-B - Résistance Pull-Down / Pull-Up

Circuit avec 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 $10\text{ k}\Omega$ (soit $10\,000\,\Omega$), 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 appliqué sur l'entrée. Le courant va prendre le chemin le plus simple, soit par la résistance de $220\,\Omega$ 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 Volt, et donc pas de signal parasite à l'entrée de D4.

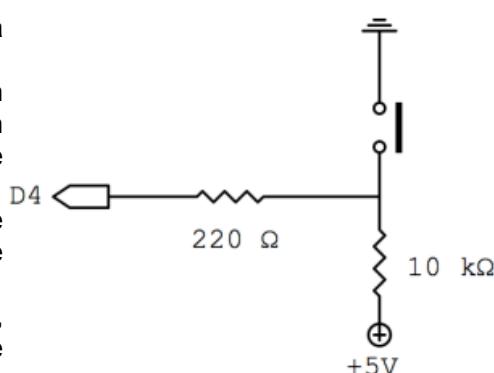
Circuit avec une résistance pull-up

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 $+5\text{ V}$.

En effet, lorsque le circuit est ouvert, une tension de $+5\text{ V}$ 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 ?

À 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 ;

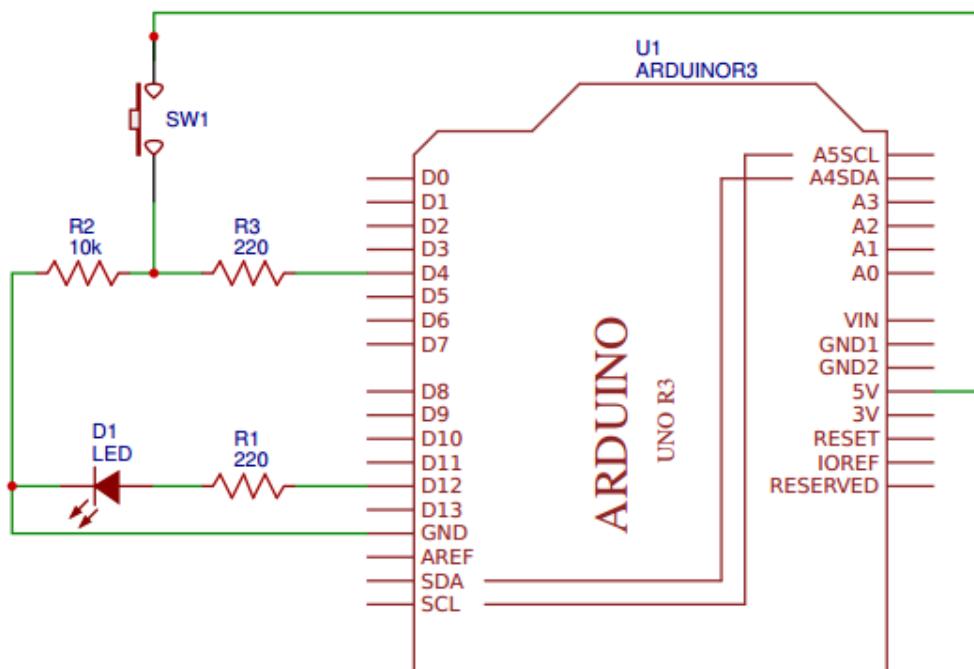
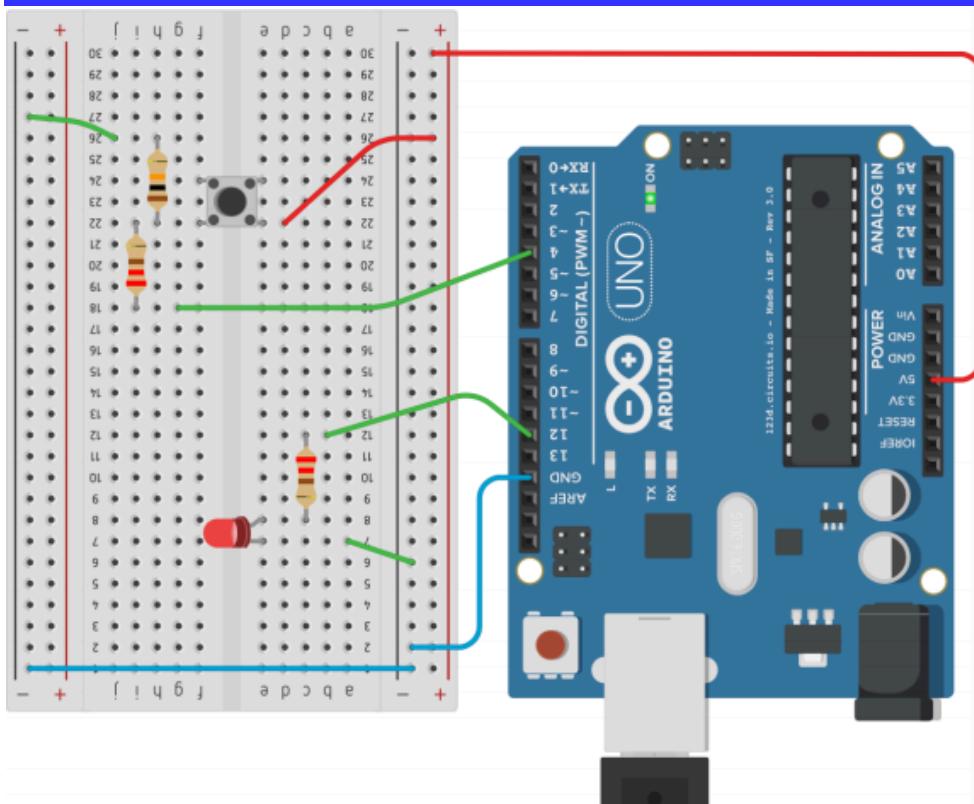
avec une résistance *pull-up*, par défaut, l'entrée sur la broche est égale à 1.

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

XVI-C - 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 470 Ω ;
- 1 résistance de 1 k à 10 k Ω ;
- 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

```
1. /*
2.   Code 10 - Edurobot.ch, destiné à l'Arduino
3.   Allume LED en fonction de l'état du bouton poussoir
4. */
5.
6. // On déclare les variables
7. const int bouton = 4; // la broche 4 devient bouton
8. const int led = 12; // la broche 12 devient led
9.
10. void setup()
11. {
12.     pinMode(bouton, INPUT); // Initialise la broche 4 comme entrée
13.     pinMode(led, OUTPUT); // Initialise la broche 12 comme sortie
14.     Serial.begin(9600); // Ouvre le port série à 9600 bauds
15. }
16.
17. void loop()
18. {
19.     // Si bouton poussoir appuyé...
20.     if (digitalRead(bouton) == 1) // teste si le bouton a une valeur de 1
21.         // ...on allume la LED
22.     {
23.         digitalWrite(led, HIGH); // allume la LED
24.     }
25.     // Sinon...
26.     else
27.         // teste si le bouton a une valeur de 0
28.         // ...on éteint la LED
29.     {
30.         digitalWrite(led, LOW); // éteint la LED
31.     }
32. }
```

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 :

```
7. const int bouton = 4; // la broche 4 devient bouton
8. 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.

```
10. void setup() {
11.     // Initialise la broche 4 comme entrée
12.     pinMode(bouton, INPUT);
13. }
```

```

14.     // Initialise la broche 12 comme sortie
15.     pinMode(led, OUTPUT);
16.
17.     // Ouvre le port série à 9600 bauds
18.     Serial.begin(9600);
19. }
```

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 0 V), il stockera une valeur de 0.

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

```

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

doit se comprendre comme :

```

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

et les lignes :

```

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

doivent se comprendre comme :

```

1. sinon (c'est-à-dire: la broche 4 a une valeur égale à 0)
2. 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>



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 : Un code plus élégant

```

1. /*
2.     Code 11 - Edurobot.ch, destiné à l'Arduino
3.     Allume LED en fonction de l'état du bouton poussoir
4. */
```

Code 11 : Un code plus élégant

```

5.
6. const int bouton = 4;      // la broche 4 devient bouton
7. const int led = 12;        // la broche 12 devient led
8. int etatbouton;          // variable qui enregistre l'état du bouton
9.
10. void setup()
11. {
12.     pinMode(bouton, INPUT); // Initialise le bouton comme entrée
13.     pinMode(led, OUTPUT);   // Initialise la led comme sortie
14.     etatbouton = LOW;       // Initialise l'état du bouton comme relâché
15.     Serial.begin(9600);    // Ouvre le port série à 9600 bauds
16. }
17.
18. void loop()
19. {
20.     etatbouton = digitalRead(bouton); // On mémorise l'état du bouton
21.     if(etatbouton == LOW) // teste si le bouton a un niveau logique BAS
22.     {
23.         digitalWrite(led,LOW); // la LED reste éteinte
24.     }
25.     else // teste si le bouton a un niveau logique différent de BAS (donc HAUT)
26.     {
27.         digitalWrite(led,HIGH); // le bouton est appuyé, la LED est allumée
28.     }
29. }
```

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

```
8. int etatbouton;
```

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

```
20. 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 :

```

21. if(etatbouton == LOW) // teste si le bouton a un niveau logique BAS
22. {
23.     digitalWrite(led,LOW); // la LED reste éteinte
24. }
25. else // teste si le bouton a un niveau logique différent de BAS (donc HAUT)
26. {
27.     digitalWrite(led,HIGH); // le bouton est appuyé, la LED est allumée
28. }
```

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

XVI-D - 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.

Code 11 version 2

```
/*
  Code 11, version 2 - Edurobot.ch, destiné à l'Arduino
  Faire clignoter la LED quand on appuie sur le 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
        delay(500); // attente de 0,5 seconde
        digitalWrite(led, LOW); // la LED est éteinte
        delay(500); // attente de 0,5 seconde
    }
}
```

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.

Code 11 version 3

```
/*
  Code 11, version 3 - Edurobot.ch, destiné à l'Arduino
  Faire clignoter la LED quand on appuie sur le 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, HIGH); // la LED est allumée
    }
    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
        delay(500); // attente de 0,5 seconde
    }
}
```

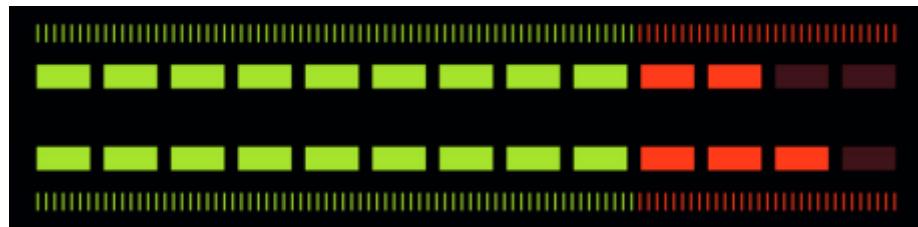
Code 11 version 3

```

        digitalWrite(led, LOW); // la LED est éteinte
        delay(500);           // attente de 0,5 seconde
    }
}
    
```

XVI-E - 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 LED alignées en guise d'affichage.



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

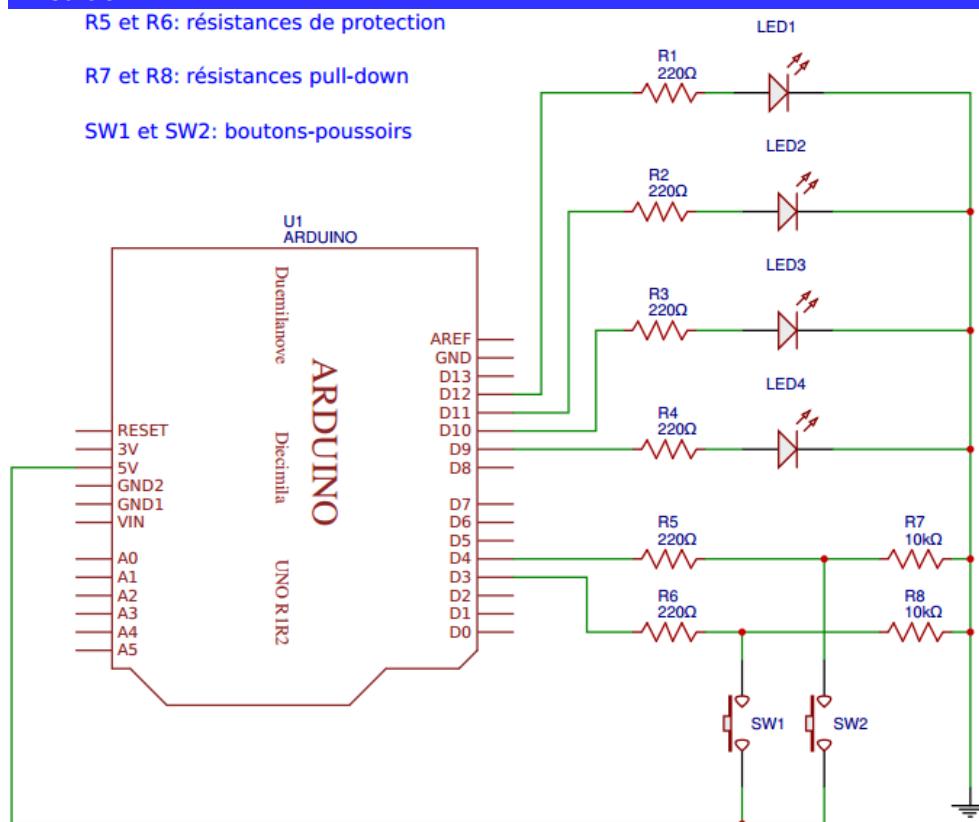
Voici le schéma du circuit à monter :

Circuit 8

R5 et R6: résistances de protection

R7 et R8: résistances pull-down

SW1 et SW2: boutons-poussoirs

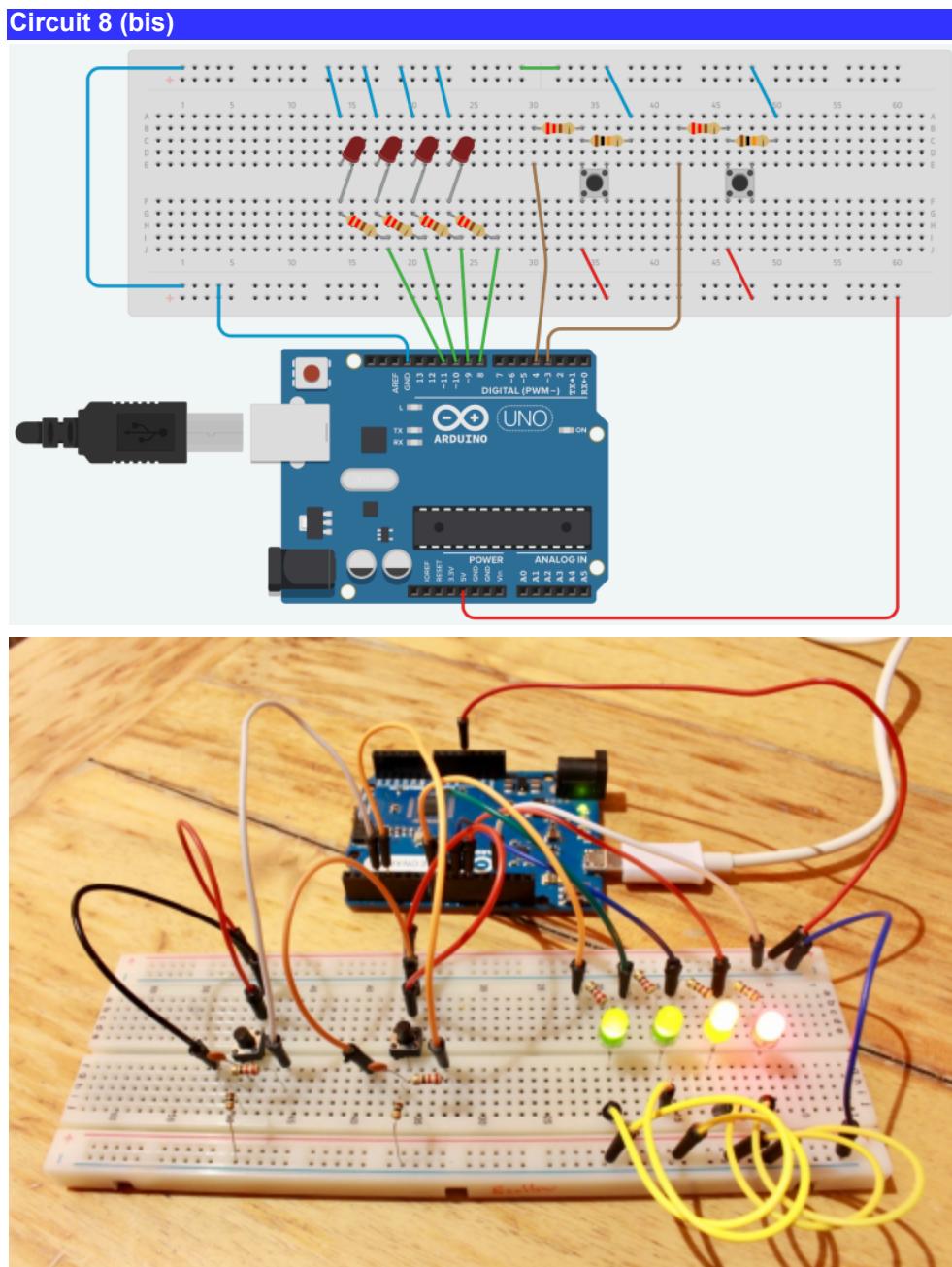

Liste des composants :

- 4 LED au choix ;
- 6 résistances de 220 Ω ;

- 2 résistances de 1 à 10 kΩ ;
- 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 électrique ! Dans le cas contraire, voici le montage à réaliser :



Code 12 : le bargraphe

```

1. /*
2.  Code 12 - Edurobot.ch, destiné à l'Arduino
3.  Le bargraphe
4.  L'objectif est de réaliser un bargraphe, avec 4 LED et deux boutons-poussoirs: un pour
5.  incrémenter le nombre de LED allumées, l'autre pour le décrémenter.
6. */
7.

```

Code 12 : le bargraphe

```

8. /*
9. Déclaration des constantes pour les noms des broches
10. */
11. const int btn_minus = 3; // Bouton 1 pour décrémenter le nombre de LED allumées
12. const int btn_plus = 4; // Bouton 2 pour incrémenter le nombre de LED allumées
13. const int led_0 = 8; // Led 0
14. const int led_1 = 9; // Led 1
15. const int led_2 = 10; // Led 2
16. const int led_3 = 11; // Led 3
17.
18. /*
19. Déclaration des variables utilisées pour le comptage et le décomptage
20. */
21. int nombre_led = 0; // le nombre qui sera incrémenté et décrémenté
22. int etat_bouton; // lecture de l'état des boutons (un seul à la fois, mais une variable suffit;
23. // en effet, il n'est pas prévu d'appuyer sur les deux boutons simultanément)
24. int memoire_plus = LOW; // état relâché par défaut pour le bouton 2
25. int memoire_minus = LOW; // état relâché par défaut pour le bouton 1
26.
27. /*
28. Initialisation des broches en entrée/sortie : entrées pour les boutons, sorties pour les LED
29. */
30. void setup()
31. {
32.     pinMode(btn_plus, INPUT);
33.     pinMode(btn_minus, INPUT);
34.     pinMode(led_0, OUTPUT);
35.     pinMode(led_1, OUTPUT);
36.     pinMode(led_2, OUTPUT);
37.     pinMode(led_3, OUTPUT);
38. }
39.
40. /*
41. Et c'est parti pour le programme !
42. */
43. void loop()
44. {
45.     // Lecture de l'état du bouton d'incrémantation
46.     // (on lit l'état du btn_plus et on l'inscrit dans la variable etat_bouton)
47.     etat_bouton = digitalRead(btn_plus);
48.     // Si le bouton a un état différent de celui enregistré ET que cet état est "appuyé"
49.     if((etat_bouton != memoire_plus) && (etat_bouton == HIGH))
50.     {
51.         nombre_led++; // on incrémente la variable qui indique combien de LED devront s'allumer
52.     }
53.
54.     memoire_plus = etat_bouton; // on enregistre l'état du bouton pour le tour suivant
55.
56.     // et maintenant pareil pour le bouton qui décrémente
57.     etat_bouton = digitalRead(btn_minus); //lecture de son état
58.
59.     // Si le bouton a un état différent de celui enregistré ET que cet état est "appuyé"
60.     if((etat_bouton != memoire_minus) && (etat_bouton == HIGH))
61.     {
62.         nombre_led--; // on décrémente la valeur de nombre_led
63.     }
64.     memoire_minus = etat_bouton; // on enregistre l'état du bouton pour le tour suivant
65.
66.     // on applique des limites au nombre pour ne pas dépasser 4 ou 0 (puisque'on a 4 LED)
67.     if(nombre_led > 4)
68.     {
69.         nombre_led = 4;
70.     }
71.     if(nombre_led < 0)
72.     {
73.         nombre_led = 0;
74.     }
75.

```

Code 12 : le bargraphe

```

76.     // On crée une fonction affiche() pour l'affichage du résultat
77.     // on lui envoie alors en paramètre la valeur du nombre de LED à éclairer
78.     affiche(nombre_led);
79. }
80.
81. void affiche(int valeur_recue)
82. {
83.     // on éteint toutes les LED
84.     digitalWrite(led_0, LOW);
85.     digitalWrite(led_1, LOW);
86.     digitalWrite(led_2, LOW);
87.     digitalWrite(led_3, LOW);
88.
89.     // Puis on les allume une à une
90.     if(valeur_recue >= 1) // "si la valeur reçue est plus grande ou égale à 1..."
91.     {
92.         digitalWrite(led_0, HIGH); // "on allume la LED 0
93.     }
94.     if(valeur_recue >= 2) // "si la valeur reçue est plus grande ou égale à 2..."
95.     {
96.         digitalWrite(led_1, HIGH); // "on allume la LED 1 (sous-entendu que la LED 0 est
97.                               // puisque la valeur est plus grande que 1)
98.     }
99.     if(valeur_recue >= 3) // "si la valeur reçue est plus grande ou égale à 3...
100.    {
101.        digitalWrite(led_2, HIGH); // "on allume la LED 2
102.    }
103.    if(valeur_recue >= 4) // "si la valeur reçue est plus grande ou égale à 4...
104.    {
105.        digitalWrite(led_3, HIGH); // "on allume la LED 3
106.    }
107. }
```

Vidéo

Vidéo de présentation du bargraphe : <https://www.scolcast.ch/episode/arduino-lecole-le-barregraphe-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 LED sont connectées aux broches 8 à 11.

```

8. /*
9. Déclaration des constantes pour les noms des broches
10. */
11. const int btn_minus = 3; // Bouton 1 pour décrémenter le nombre de LED allumées
12. const int btn_plus = 4; // Bouton 2 pour incrémenter le nombre de LED allumés
13. const int led_0 = 8; // Led 0
14. const int led_1 = 9; // Led 1
15. const int led_2 = 10; // Led 2
16. const int led_3 = 11; // Led 3
```

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

```

18. /*
19. Déclaration des variables utilisées pour le comptage et le décomptage
20. */
21. int nombre_led = 0; // le nombre qui sera incrémenté et décrémenté
22. int etat_bouton; // lecture de l'état des boutons (un seul à la fois, mais une variable
    suffit;
23. // en effet, il n'est pas prévu d'appuyer sur les deux boutons
    simultanément)
24. int memoire_plus = LOW; // état relâché par défaut pour le bouton 2
25. 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 LED).

```

27. /*
28. Initialisation des broches en entrée/sortie : entrées pour les boutons, sorties pour les LED
29. */
30. void setup()
31. {
32.     pinMode(btn_plus, INPUT);
33.     pinMode(btn_minus, INPUT);
34.     pinMode(led_0, OUTPUT);
35.     pinMode(led_1, OUTPUT);
36.     pinMode(led_2, OUTPUT);
37.     pinMode(led_3, OUTPUT);
38. }
```

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`.

```

43. void loop()
44. {
45.     // Lecture de l'état du bouton d'incrémentation
46.     // (on lit l'état du btn_plus et on l'inscrit dans la variable etat_bouton)
47.     etat_bouton = digitalRead(btn_plus);
```

Maintenant, ça devient sérieux. Regardons ce qui suit :

```

48.     // Si le bouton a un état différent de celui enregistré ET que cet état est "appuyé"
49.     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.

```

50.     {
51.         nombre_led++; // on incrémente la variable qui indique combien de LED devront
52.         s'allumer
53.     }
```

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

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

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

```

56.     // et maintenant pareil pour le bouton qui décrémente
57.     etat_bouton = digitalRead(btn_minus); //lecture de son état
58.
59.     // Si le bouton a un état différent de celui enregistré ET que cet état est "appuyé"
60.     if((etat_bouton != memoire_minus) && (etat_bouton == HIGH))
61.     {
62.         nombre_led--; // on décrémente la valeur de nombre_led
63.     }
64.     memoire_minus = etat_bouton; // on enregistre l'état du bouton pour le tour suivant
```

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

```

66. // on applique des limites au nombre pour ne pas dépasser 4 ou 0 (puisque'on a 4 LED)
67. if(nombre_led > 4)
68. {
69.     nombre_led = 4;
70. }
```

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

Même chose maintenant pour le bouton 1.

```

71. if(nombre_led < 0)
72. {
73.     nombre_led = 0;
74. }
```

Maintenant, nous devons gérer l'allumage des LED. 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.

```

76. // On crée une fonction affiche() pour l'affichage du résultat
77. // on lui envoie alors en paramètre la valeur du nombre de LED à éclairer
78. affiche(nombre_led);
79. }
80.
81. void affiche(int valeur_recue)
```

On commence d'abord par éteindre toutes les LED.

```

82. {
83.     // on éteint toutes les LED
84.     digitalWrite(led_0, LOW);
85.     digitalWrite(led_1, LOW);
86.     digitalWrite(led_2, LOW);
87.     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 LED :

```

89. // Puis on les allume une à une
90. if(valeur_recue >= 1) // "si la valeur reçue est plus grande ou égale à 1..."
91. {
92.     digitalWrite(led_0, HIGH); // "on allume la LED 0
93. }
94. if(valeur_recue >= 2) // "si la valeur reçue est plus grande ou égale à 2..."
95. {
96.     digitalWrite(led_1, HIGH); // "on allume la LED 1 (sous-entendu que la LED 0 est
allumée,
97.                         // puisque la valeur est plus grande que 1)
98. }
99. if(valeur_recue >= 3) // "si la valeur reçue est plus grande ou égale à 3...
100. {
101.     digitalWrite(led_2, HIGH); // "on allume la LED 2
102. }
103. if(valeur_recue >= 4) // "si la valeur reçue est plus grande ou égale à 4...
104. {
105.     digitalWrite(led_3, HIGH); // "on allume la LED 3
106. }
107. }
```

Le symbole `>=` signifie : ...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 :

```

103.     if(valeur_recue >= 4) // "si la valeur reçue est plus grande ou égale à 4..."
104. {
105.     digitalWrite(led_3, HIGH); // "on allume la LED 3
106. }

```

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

XVI-F - 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 LED s'allument ou s'éteignent pour une seule 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 0 V à 5 V 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 LED 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 (10 nF).

Qu'est-ce qu'un condensateur ?

Le condensateur est un composant électronique passif, comme les résistances. Il a pour faculté d'emmageriner une charge électrique, avant de pouvoir la restituer en cas de baisse de la tension. On utilise ainsi les condensateurs 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).



Symbole du condensateur

Circuit 9

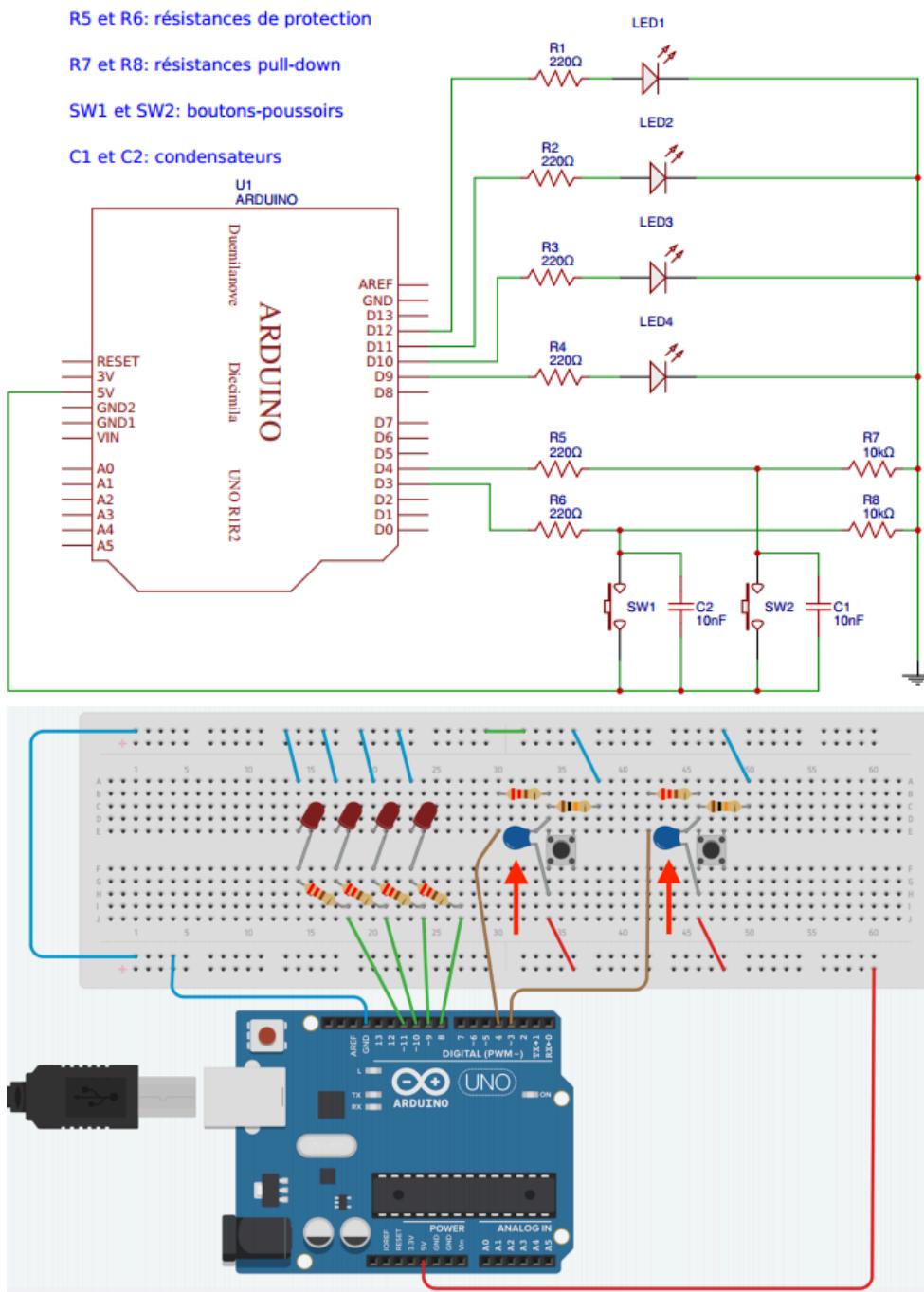
LED1 à LED4: à choix

R5 et R6: résistances de protection

B7 et B8: résistances pull-down

SW1 et SW2: boutons-poussoirs

C1 et C2: condensateurs

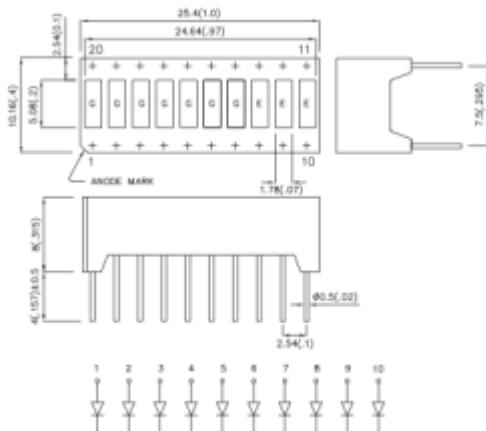


Comme on peut le voir sur le circuit ci-dessus, deux condensateurs de faible capacité ont été ajoutés en parallèle aux boutons-poussoirs. Si la parade n'est pas absolue, cela permet de considérablement augmenter la précision des boutons.

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

XVI-G - Variation : le bargraph à 10 LED

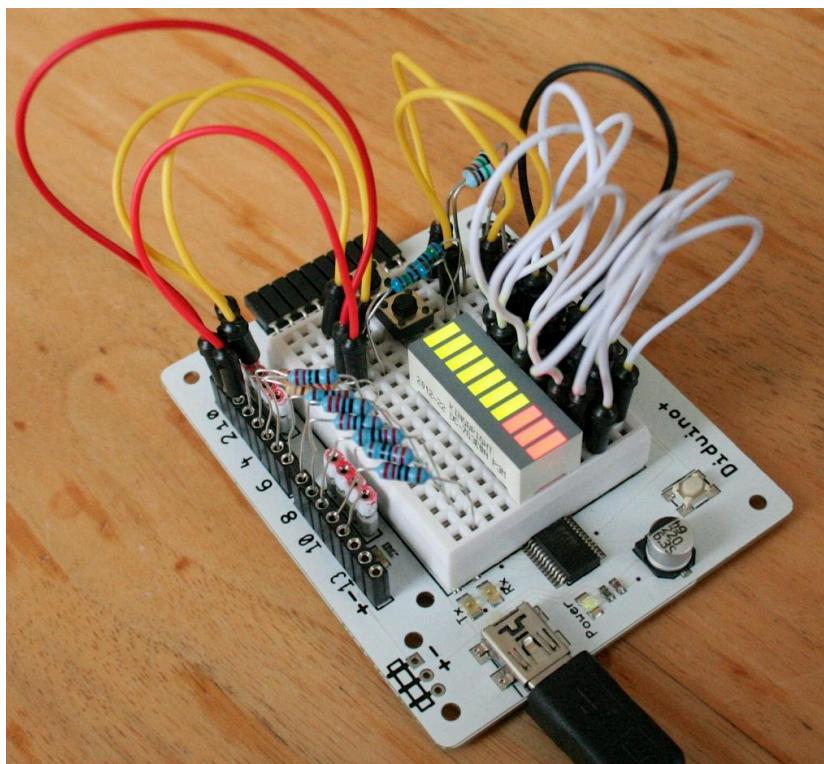
Il existe des bargraphes intégrant 10 LED. On peut les utiliser en remplacement des 4 LED du précédent montage.



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 LED. Un exemple se trouve ici :



Code 13 : le bargraphe à 10 LED

```

1. /*
2.   Code 13 - Edurobot.ch, destiné à l'Arduino
3.   Le bargraphe
4.   L'objectif est de réaliser un bargraphe, avec deux boutons-
5.   poussoirs : un pour incrémenter le nombre de LED allumées,
6.   l'autre pour le décrémenter.
7.   Le bargraphe est composé de 10 LED.
8. */
9.
10. /*

```

Code 13 : le bargraphe à 10 LED

```

11. Déclaration des constantes pour les noms des broches
12. /*
13. const int btn_minus = 2; // Bouton 1 pour décrémenter le nombre de LED allumées
14. const int btn_plus = 3; // Bouton 2 pour incrémenter le nombre de LED allumées
15. const int led_0 = 4; // Led 0
16. const int led_1 = 5; // Led 1
17. const int led_2 = 6; // Led 2
18. const int led_3 = 7; // Led 3
19. const int led_4 = 8; // Led 4
20. const int led_5 = 9; // Led 5
21. const int led_6 = 10; // Led 6
22. const int led_7 = 11; // Led 7
23. const int led_8 = 12; // Led 8
24. const int led_9 = 13; // Led 9
25.
26. /*
27. Déclaration des variables utilisées pour le comptage et le décomptage
28. /*
29. int nombre_led = 0; // le nombre qui sera incrémenté et décrémenté
30. int etat_bouton; // lecture de l'état des boutons (un seul à la fois, mais une variable suffit;
31. // en effet, il n'est pas prévu d'appuyer sur les deux boutons simultanément)
32. int memoire_plus = LOW; // état relâché par défaut pour le bouton 2
33. int memoire_minus = LOW; // état relâché par défaut pour le bouton 1
34.
35. /*
36. Initialisation des broches en entrée/sortie : entrées pour les boutons, sorties pour les LED
37. /*
38. void setup()
39. {
40.     pinMode(btn_plus, INPUT);
41.     pinMode(btn_minus, INPUT);
42.     pinMode(led_0, OUTPUT);
43.     pinMode(led_1, OUTPUT);
44.     pinMode(led_2, OUTPUT);
45.     pinMode(led_3, OUTPUT);
46.     pinMode(led_4, OUTPUT);
47.     pinMode(led_5, OUTPUT);
48.     pinMode(led_6, OUTPUT);
49.     pinMode(led_7, OUTPUT);
50.     pinMode(led_8, OUTPUT);
51.     pinMode(led_9, OUTPUT);
52. }
53.
54. void loop()
55. {
56.     // lecture de l'état du bouton d'incrémantation (on lit l'état du btn_plus
57.     // et on l'inscrit dans la variable etat_bouton)
58.     etat_bouton = digitalRead(btn_plus);
59.
60.     // Si le bouton a un état différent de celui enregistré ET que cet état est "appuyé"
61.     if ((etat_bouton != memoire_plus) && (etat_bouton == HIGH))
62.     {
63.         nombre_led++; //on incrémente la variable qui indique combien de LED devront s'allumer
64.     }
65.     memoire_plus = etat_bouton; //on enregistre l'état du bouton pour le tour suivant
66.
67.     // et maintenant pareil pour le bouton qui décrémente
68.     etat_bouton = digitalRead(btn_minus); //lecture de son état
69.
70.     // Si le bouton a un état différent de celui enregistré ET que cet état est "appuyé"
71.     if ((etat_bouton != memoire_minus) && (etat_bouton == HIGH))
72.     {
73.         nombre_led--; //on décrémente la valeur de nombre_led
74.     }
75.     memoire_minus = etat_bouton; //on enregistre l'état du bouton pour le tour suivant
76.
77.     // on applique des limites au nombre pour ne pas dépasser 10 ou 0 (puisque'on a 10 LED)
78.     if (nombre_led > 10)

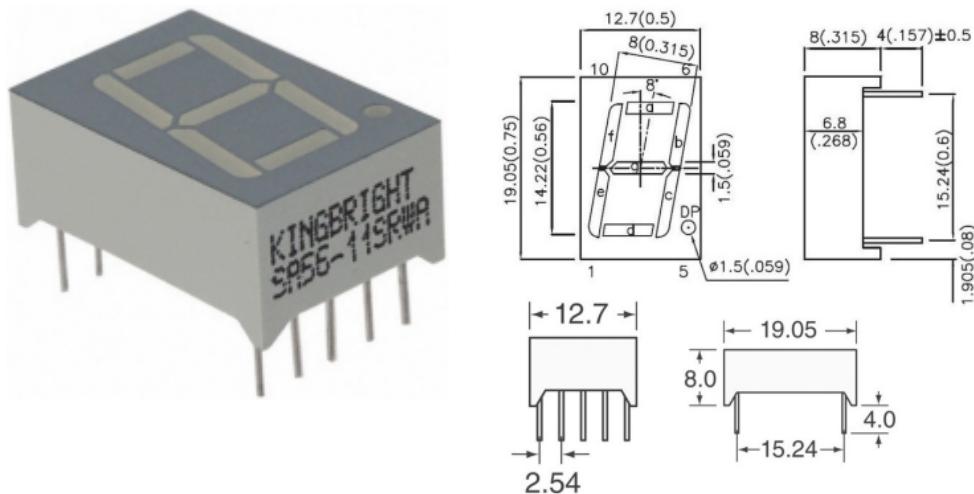
```

Code 13 : le bargraphe à 10 LED

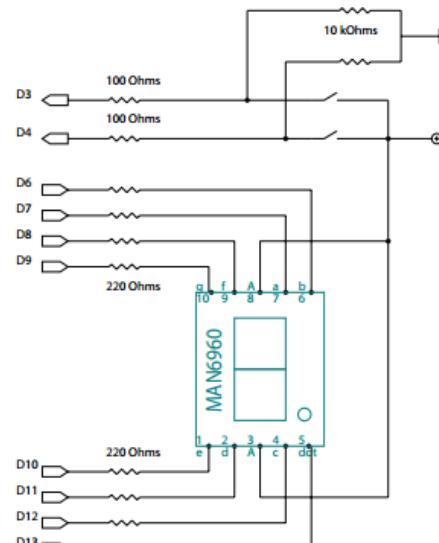
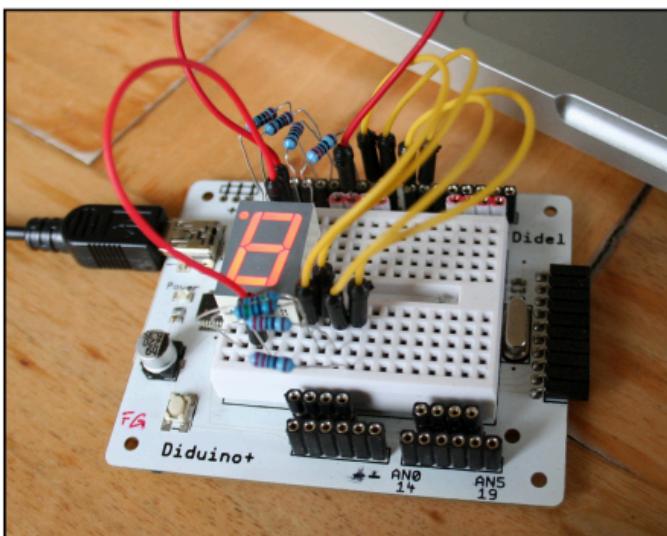
```

79. {
80.     nombre_led = 10;
81. }
82. if (nombre_led < 0)
83. {
84.     nombre_led = 0;
85. }
86.
87. // On crée une fonction affiche() pour l'affichage du résultat
88. // on lui envoie alors en paramètre la valeur du nombre de LED à éclairer
89. affiche(nombre_led);
90. }
91.
92. void affiche(int valeur_recue)
93. {
94.     // on éteint toutes les LED
95.     digitalWrite(led_0, LOW);
96.     digitalWrite(led_1, LOW);
97.     digitalWrite(led_2, LOW);
98.     digitalWrite(led_3, LOW);
99.     digitalWrite(led_4, LOW);
100.    digitalWrite(led_5, LOW);
101.    digitalWrite(led_6, LOW);
102.    digitalWrite(led_7, LOW);
103.    digitalWrite(led_8, LOW);
104.    digitalWrite(led_9, LOW);
105.
106. //Puis on les allume une à une
107. if (valeur_recue >= 1) // "si la valeur reçue est plus grande ou égale à 1..."
108. {
109.     digitalWrite(led_0, HIGH); // "on allume la LED 0
110. }
111. if (valeur_recue >= 2) // "si la valeur reçue est plus grande ou égale à 2..."
112. {
113.     digitalWrite(led_1, HIGH); // "on allume la LED 1 (sous-entendu que la LED 0 est
114.                                   allumée,
115.                                   // puisque la valeur est plus grande que 1)
116.     if (valeur_recue >= 3) // "si la valeur reçue est plus grande ou égale à 3..."
117.     {
118.         digitalWrite(led_2, HIGH); // "on allume la LED 2
119.     }
120.     if (valeur_recue >= 4) // "si la valeur reçue est plus grande ou égale à 4..."
121.     {
122.         digitalWrite(led_3, HIGH); // "on allume la LED 3
123.     }
124.     if (valeur_recue >= 5) // "si la valeur reçue est plus grande ou égale à 5..."
125.     {
126.         digitalWrite(led_4, HIGH); // "on allume la LED 4
127.     }
128.     if (valeur_recue >= 6) // "si la valeur reçue est plus grande ou égale à 6..."
129.     {
130.         digitalWrite(led_5, HIGH); // "on allume la LED 5
131.     }
132.     if (valeur_recue >= 7) // "si la valeur reçue est plus grande ou égale à 7..."
133.     {
134.         digitalWrite(led_6, HIGH); // "on allume la LED 6
135.     }
136.     if (valeur_recue >= 8) // "si la valeur reçue est plus grande ou égale à 8..."
137.     {
138.         digitalWrite(led_7, HIGH); // "on allume la LED 7
139.     }
140.     if (valeur_recue >= 9) // "si la valeur reçue est plus grande ou égale à 9..."
141.     {
142.         digitalWrite(led_8, HIGH); // "on allume la LED 8
143.     }
144.     if (valeur_recue >= 10) // "si la valeur reçue est plus grande ou égale à 10..."
145.     {
146.         digitalWrite(led_9, HIGH); // "on allume la LED 9
147.     }
148. }
```

XVI-H - 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 (8) .



Dans le cas d'une anode commune, cette dernière est branchée sur le +5 V. Les cathodes sont branchées sur les broches.

Au niveau du code, cela implique que les LED sont allumées 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.

Identification de la position des LED

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

Code 14 : l'afficheur 7 ou 8 segments

```

1. /*
2.  Code 14 - Edurobot.ch, destiné à l'Arduino
3.  L'afficheur 7 ou 8 segments
4.  L'objectif est d'afficher des chiffres sur un afficheur 7 segments (7 digits).

```

Code 14 : l'afficheur 7 ou 8 segments

```

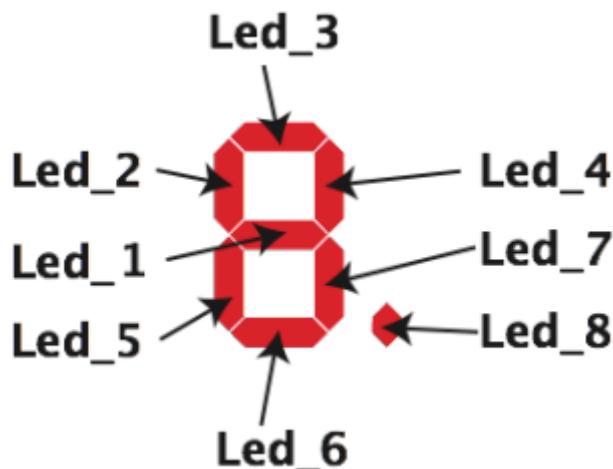
5.   Ce code a pour objectif d'allumer toutes les LED puis l'une après l'autre
6.   pour identifier leur position.
7. */
8.
9. /*
10. Déclaration des constantes pour les noms des broches
11. */
12. const int led_1 = 6; // Led 1
13. const int led_2 = 7; // Led 2
14. const int led_3 = 8; // Led 3
15. const int led_4 = 9;
16. const int led_5 = 10;
17. const int led_6 = 11;
18. const int led_7 = 12;
19. const int led_8 = 13;
20.
21. void setup()
22. {
23.     pinMode(led_1, OUTPUT);
24.     pinMode(led_2, OUTPUT);
25.     pinMode(led_3, OUTPUT);
26.     pinMode(led_4, OUTPUT);
27.     pinMode(led_5, OUTPUT);
28.     pinMode(led_6, OUTPUT);
29.     pinMode(led_7, OUTPUT);
30.     pinMode(led_8, OUTPUT);
31. }
32.
33. /*
34. Et c'est parti pour le programme!
35. */
36. void loop()
37. {
38.     // on allume toutes les LED
39.     digitalWrite(led_1, LOW);
40.     digitalWrite(led_2, LOW);
41.     digitalWrite(led_3, LOW);
42.     digitalWrite(led_4, LOW);
43.     digitalWrite(led_5, LOW);
44.     digitalWrite(led_6, LOW);
45.     digitalWrite(led_7, LOW);
46.     digitalWrite(led_8, LOW);
47.     delay(1500);
48.
49.     // on éteint toutes les LED
50.     digitalWrite(led_1, HIGH);
51.     digitalWrite(led_2, HIGH);
52.     digitalWrite(led_3, HIGH);
53.     digitalWrite(led_4, HIGH);
54.     digitalWrite(led_5, HIGH);
55.     digitalWrite(led_6, HIGH);
56.     digitalWrite(led_7, HIGH);
57.     digitalWrite(led_8, HIGH);
58.     delay(500);
59.
60.     // on allume une diode après l'autre pour identifier sa position
61.     digitalWrite(led_1, LOW);
62.     delay(1500);
63.
64.     digitalWrite(led_1, HIGH);
65.     digitalWrite(led_2, LOW);
66.     delay(1500);
67.
68.     digitalWrite(led_2, HIGH);
69.     digitalWrite(led_3, LOW);
70.     delay(1500);
71.
72.     digitalWrite(led_3, HIGH);
73.     digitalWrite(led_4, LOW);
74.     delay(1500);
75.
```

Code 14 : l'afficheur 7 ou 8 segments

```

76.     digitalWrite(led_4, HIGH);
77.     digitalWrite(led_5, LOW);
78.     delay(1500);
79.
80.     digitalWrite(led_5, HIGH);
81.     digitalWrite(led_6, LOW);
82.     delay(1500);
83.
84.     digitalWrite(led_6, HIGH);
85.     digitalWrite(led_7, LOW);
86.     delay(1500);
87.
88.     digitalWrite(led_7, HIGH);
89.     digitalWrite(led_8, LOW);
90.     delay(1500);
91. }
```

L'objectif est d'identifier chaque LED sur le schéma ci-dessous :



Voici donc, dans notre cas, les LED 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 :

Code 16

```

/*
Code 16 - Edurobot.ch, destiné au Diduino
Apprendre à compter
```

Code 16

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.

*/

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

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

/*
 Initialisation des pattes en entrée/sortie : entrées pour les boutons, sorties pour les LED
 */

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 de 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 LED devront s'allumer
    }

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

Code 16

```

//et maintenant pareil pour le bouton qui décrémente
etat_bouton = digitalRead(btn_minus); //lecture de son état

//Si le bouton a un état différent de 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 10 ou 0 (puisque'on a 10 LED)
if(nombre_led > 10)
{
    nombre_led = 10;
}
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 LED
    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);

    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);
    }
}

```

Code 16

```

}
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)
{

```

Code 16

```

{

  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);

}

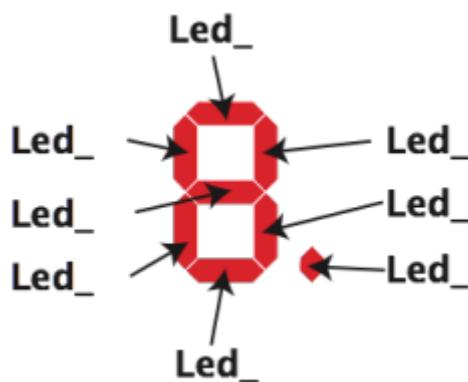
if(valeur_recue == 10)
{

  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);

}
}

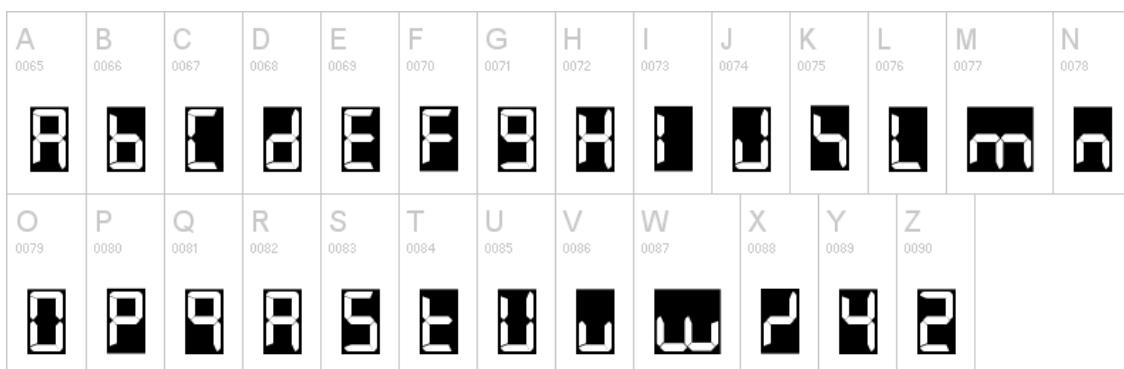
```

À ton tour :

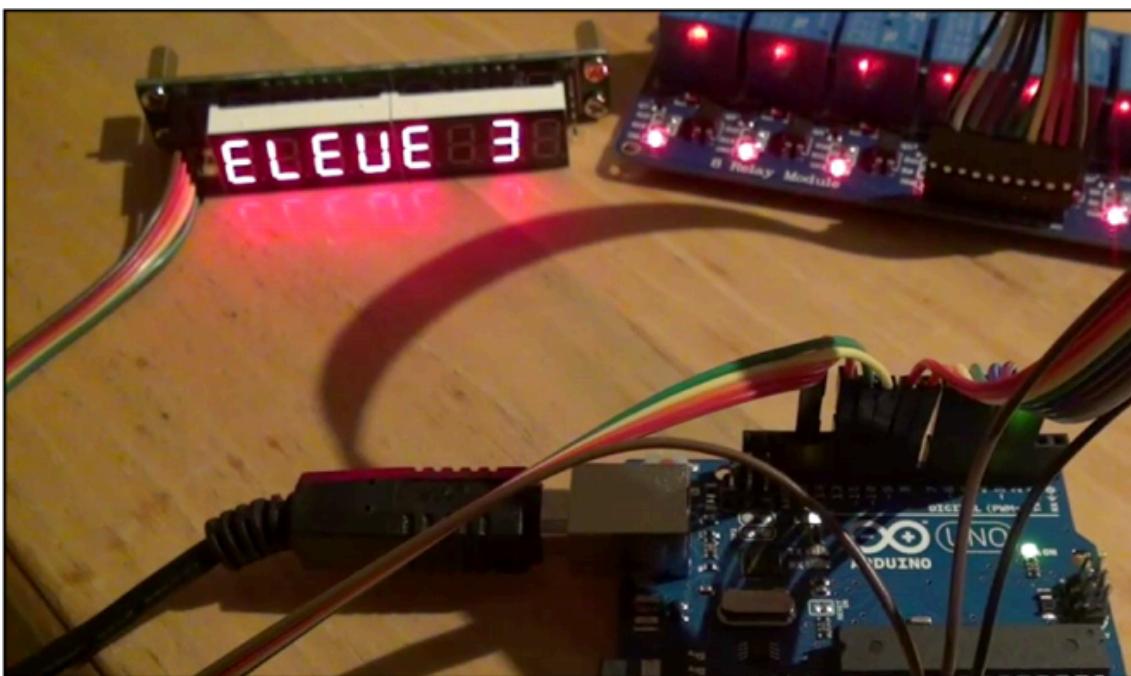


#	led_1	led_2	led_3	led_4	led_5	led_6	led_7
1							
2							
3							
4							
5							
6							
7							
8							
9							
0							

Il est aussi possible d'écrire les lettres de l'alphabet à l'aide d'un affichage 7 segments (9) :



Exemple :



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

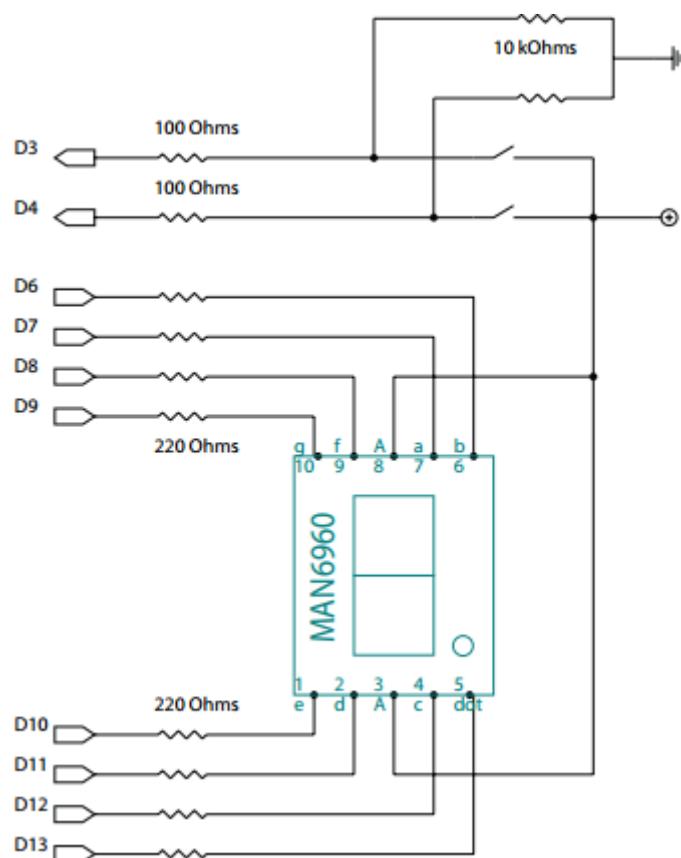
XVI-I - Synthèse : apprendre à compter

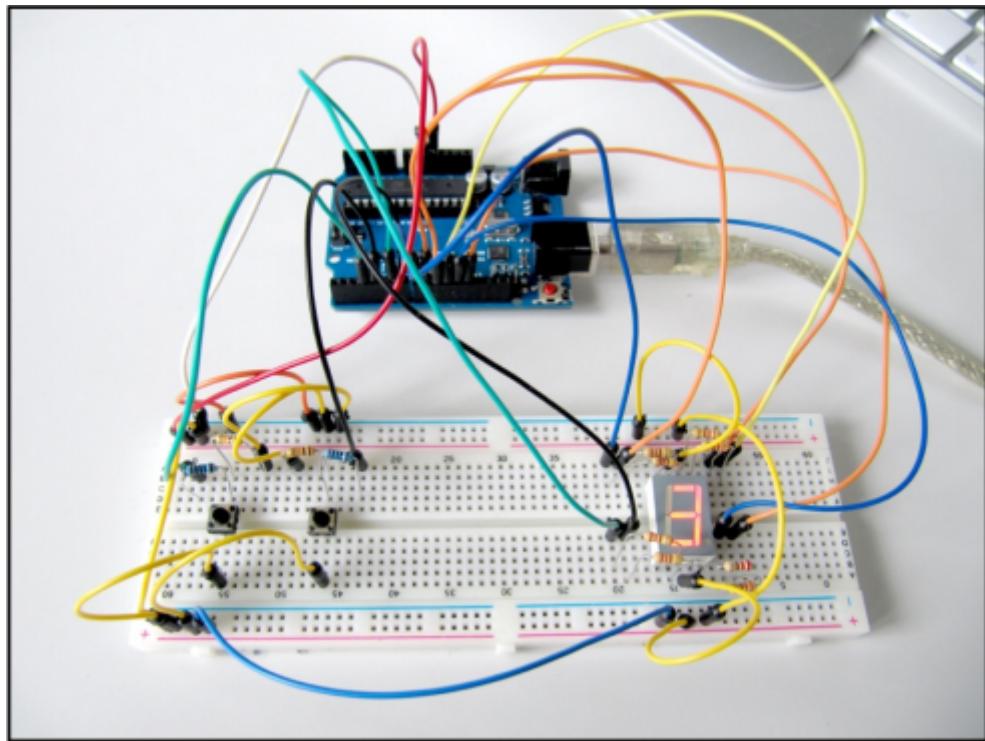
Objectif

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.

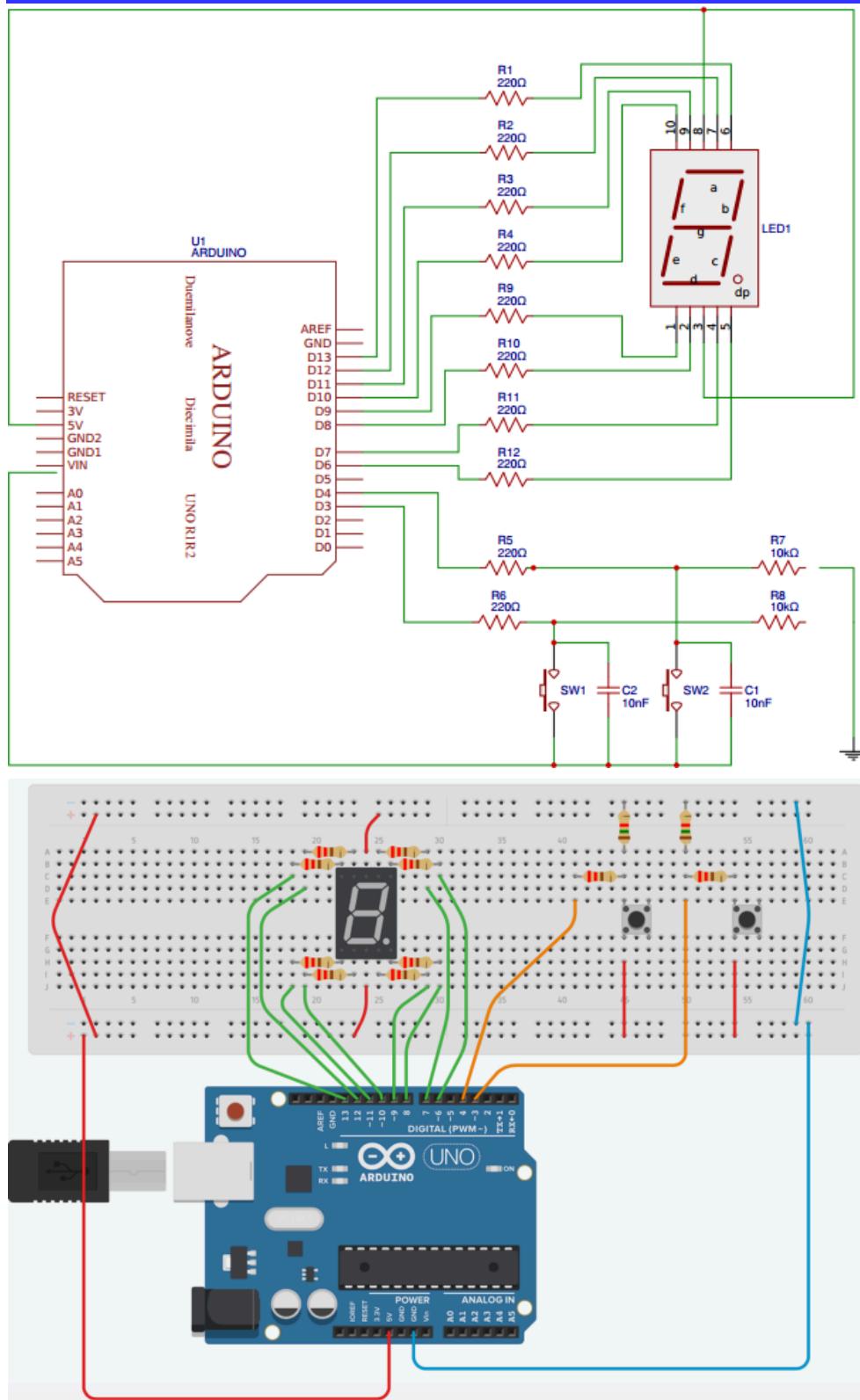
Schéma électronique du montage

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 $1\text{ k}\Omega$ ou $10\text{ k}\Omega$ et d'une résistance de protection de l'*input* de $100\ \Omega$. 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



XVI-J - Code 15 : apprendre à compter

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 LED (une LED pour 1, deux LED pour 2...), l'objectif est ici de n'allumer que les LED 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 LED allumées.

Code 15 : Apprendre à compter

```

1. /*
2.     Code 15 - Edurobot.ch, destiné au Diduino
3.     Apprendre à compter
4.     L'objectif est d'afficher des chiffres sur un afficheur 7 segments. Un bouton permet
    d'incrémenter le chiffre, l'autre de le décrémenter.
5.     On compte ainsi de 0 à 9 ou de 10 à 0.
6. */
7.
8.
9. /*
10. Déclaration des constantes pour les noms des pattes
11.*/
12. const int btn_minus = 3;      // Bouton 1 pour décrémenter les chiffres
13. const int btn_plus = 4;      // Bouton 2 pour incrémenter les chiffres
14. const int led_1 = 6;         // Led 1
15. const int led_2 = 7;         // Led 2
16. const int led_3 = 8;         // Led 3
17. const int led_4 = 9;         // Led 4
18. const int led_5 = 10;
19. const int led_6 = 11;
20. const int led_7 = 12;
21. const int led_8 = 13;
22.
23. /*
24. Déclaration des variables utilisées pour le comptage et le décomptage
25.*/
26. int nombre_led = 0;          // le nombre qui sera incrémenté et décrémenté
27. 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)
28. int memoire_plus = LOW;      // état relâché par défaut pour le bouton 2
29. int memoire_minus = LOW;    // état relâché par défaut pour le bouton 1
30.
31.
32. /*
33. Initialisation des pattes en entrée/sortie : entrées pour les boutons, sorties pour les LED
34.*/
35. void setup()
36. {
37.     pinMode(btn_plus, INPUT);
38.     pinMode(btn_minus, INPUT);
39.     pinMode(led_1, OUTPUT);
40.     pinMode(led_2, OUTPUT);
41.     pinMode(led_3, OUTPUT);
42.     pinMode(led_4, OUTPUT);
43.     pinMode(led_5, OUTPUT);
44.     pinMode(led_6, OUTPUT);
45.     pinMode(led_7, OUTPUT);
46.     pinMode(led_8, OUTPUT);
47. }
48.
49. /*
50. Et c'est parti pour le programme !
51.*/
52. void loop()
53. {
54.     // lecture de l'état du bouton d'incrémantation (on lit l'état du btn_plus et on
    l'inscrit dans la variable etat_bouton)
55.     etat_bouton = digitalRead(btn_plus);
56.
57.     // Si le bouton a un état différent de celui enregistré ET que cet état est "appuyé"
58.     if((etat_bouton != memoire_plus) && (etat_bouton == HIGH))
59.     {
60.         nombre_led++; // on incrémente la variable qui indique combien de LED devront
    s'allumer
61.     }
62.     memoire_plus = etat_bouton; // on enregistre l'état du bouton pour le tour suivant

```

Code 15 : Apprendre à compter

```

63.
64. // et maintenant pareil pour le bouton qui décrémente
65. etat_bouton = digitalRead(btn_minus); //lecture de son état
66.
67. // Si le bouton a un état différent de celui enregistré ET que cet état est "appuyé"
68. if((etat_bouton != memoire_minus) && (etat_bouton == HIGH))
69.
70.     nombre_led--; // on décrémente la valeur de nombre_led
71. }
72. memoire_minus = etat_bouton; // on enregistre l'état du bouton pour le tour suivant
73.
74. // on applique des limites au nombre pour ne pas dépasser 10 ou 0 (puisque'on a 10 LED)
75. if(nombre_led > 10)
76.
77.     nombre_led = 10;
78. }
79. if(nombre_led < 0)
80.
81.     nombre_led = 0;
82. }
83.
84.
85. // On crée une fonction affiche() pour l'affichage du résultat
86. // on lui envoie alors en paramètre la valeur du nombre de LED à éclairer
87. affiche(nombre_led);
88. }
89.
90. void affiche(int valeur_recue)
91. {
92.     // on éteint toutes les LED
93.     digitalWrite(led_1, HIGH);
94.     digitalWrite(led_2, HIGH);
95.     digitalWrite(led_3, HIGH);
96.     digitalWrite(led_4, HIGH);
97.     digitalWrite(led_5, HIGH);
98.     digitalWrite(led_6, HIGH);
99.     digitalWrite(led_7, HIGH);
100.    digitalWrite(led_8, HIGH);
101.
102.    if(valeur_recue == 1)           // "si la valeur reçue est égale à 1, on allume les
    segments pour afficher le chiffre 1"
103.    {
104.        digitalWrite(led_1, LOW);
105.        digitalWrite(led_2, HIGH);
106.        digitalWrite(led_3, HIGH);
107.        digitalWrite(led_4, LOW);
108.        digitalWrite(led_5, HIGH);
109.        digitalWrite(led_6, HIGH);
110.        digitalWrite(led_7, HIGH);
111.        digitalWrite(led_8, HIGH);
112.    }
113.
114.    if(valeur_recue == 2)           // "si la valeur reçue est égale à 2, on allume les
    segments pour afficher le chiffre 2"
115.    {
116.        digitalWrite(led_1, LOW);
117.        digitalWrite(led_2, LOW);
118.        digitalWrite(led_3, LOW);
119.        digitalWrite(led_4, HIGH);
120.        digitalWrite(led_5, LOW);
121.        digitalWrite(led_6, LOW);
122.        digitalWrite(led_7, HIGH);
123.        digitalWrite(led_8, HIGH);
124.    }
125.    if(valeur_recue == 3)           // "si la valeur reçue est égale... enfin... tu connais
    la suite...
126.    {
127.        digitalWrite(led_1, HIGH);
128.        digitalWrite(led_2, LOW);
129.        digitalWrite(led_3, LOW);
130.        digitalWrite(led_4, HIGH);

```

Code 15 : Apprendre à compter

```

131.         digitalWrite(led_5, LOW);
132.         digitalWrite(led_6, LOW);
133.         digitalWrite(led_7, LOW);
134.         digitalWrite(led_8, HIGH);
135.     }
136.     if(valeur_recue == 4)
137.     {
138.         digitalWrite(led_1, HIGH);
139.         digitalWrite(led_2, HIGH);
140.         digitalWrite(led_3, LOW);
141.         digitalWrite(led_4, LOW);
142.         digitalWrite(led_5, HIGH);
143.         digitalWrite(led_6, LOW);
144.         digitalWrite(led_7, LOW);
145.         digitalWrite(led_8, HIGH);
146.     }
147.
148.     if(valeur_recue == 5)
149.     {
150.         digitalWrite(led_1, HIGH);
151.         digitalWrite(led_2, LOW);
152.         digitalWrite(led_3, LOW);
153.         digitalWrite(led_4, LOW);
154.         digitalWrite(led_5, LOW);
155.         digitalWrite(led_6, HIGH);
156.         digitalWrite(led_7, LOW);
157.         digitalWrite(led_8, HIGH);
158.     }
159.     if(valeur_recue == 6)
160.     {
161.         digitalWrite(led_1, LOW);
162.         digitalWrite(led_2, LOW);
163.         digitalWrite(led_3, LOW);
164.         digitalWrite(led_4, LOW);
165.         digitalWrite(led_5, LOW);
166.         digitalWrite(led_6, HIGH);
167.         digitalWrite(led_7, LOW);
168.         digitalWrite(led_8, HIGH);
169.     }
170.
171.     if(valeur_recue == 7)
172.     {
173.         digitalWrite(led_1, HIGH);
174.         digitalWrite(led_2, HIGH);
175.         digitalWrite(led_3, HIGH);
176.         digitalWrite(led_4, HIGH);
177.         digitalWrite(led_5, LOW);
178.         digitalWrite(led_6, LOW);
179.         digitalWrite(led_7, LOW);
180.         digitalWrite(led_8, HIGH);
181.     }
182.
183.     if(valeur_recue == 8)
184.     {
185.         digitalWrite(led_1, LOW);
186.         digitalWrite(led_2, LOW);
187.         digitalWrite(led_3, LOW);
188.         digitalWrite(led_4, LOW);
189.         digitalWrite(led_5, LOW);
190.         digitalWrite(led_6, LOW);
191.         digitalWrite(led_7, LOW);
192.         digitalWrite(led_8, LOW);
193.     }
194.
195.     if(valeur_recue == 9)
196.     {
197.         digitalWrite(led_1, HIGH);
198.         digitalWrite(led_2, LOW);
199.         digitalWrite(led_3, LOW);
200.         digitalWrite(led_4, LOW);
201.         digitalWrite(led_5, LOW);

```

Code 15 : Apprendre à compter

```

202.     digitalWrite(led_6, LOW);
203.     digitalWrite(led_7, LOW);
204.     digitalWrite(led_8, LOW);
205. }
206.
207. if(valeur_recue == 10)
208. {
209.     digitalWrite(led_1, LOW);
210.     digitalWrite(led_2, LOW);
211.     digitalWrite(led_3, HIGH);
212.     digitalWrite(led_4, LOW);
213.     digitalWrite(led_5, LOW);
214.     digitalWrite(led_6, LOW);
215.     digitalWrite(led_7, LOW);
216.     digitalWrite(led_8, LOW);
217. }
218. }
```

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.

XVII - Projet 9 : les inputs analogiques

Un signal analogique (10) 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...

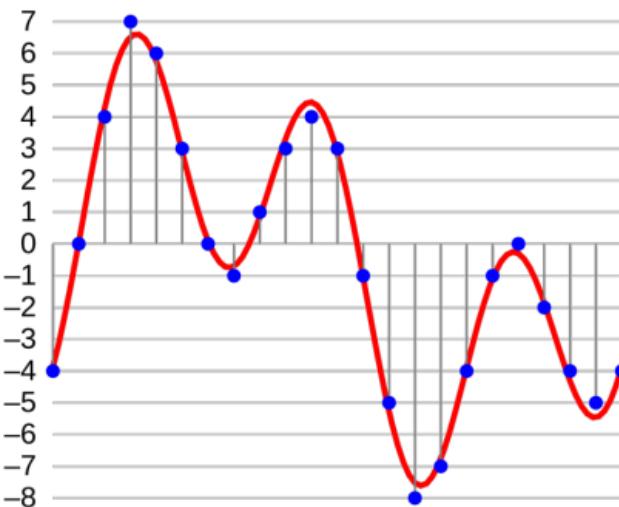
Signal numérique



Signal analogique



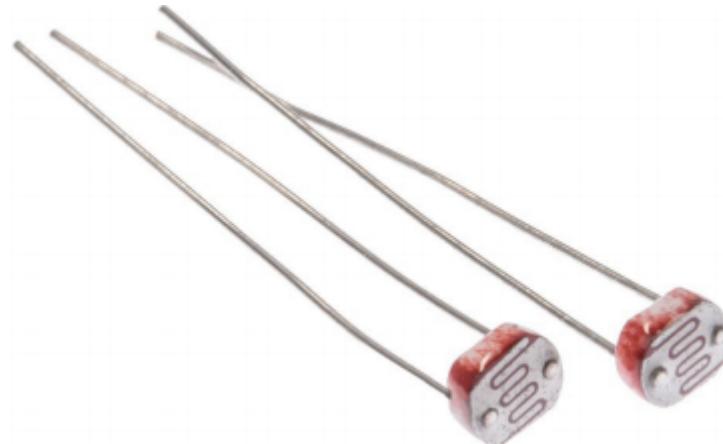
L'Arduino Uno possède six entrées analogiques, numérotées A0 à A5. En réalité, le microcontrôleur n'est pas capable de comprendre un signal analogique. Il faut donc d'abord le convertir en signal numérique par un circuit spécial appelé convertisseur analogique/numérique. Ce convertisseur va échantillonner 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 5 V, ce qui détruirait l'Arduino.



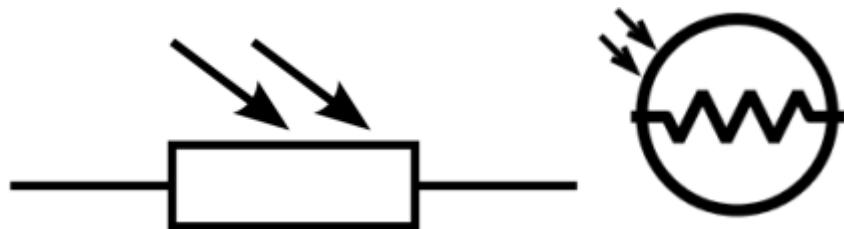
XVII-A - La photorésistance

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 :

- mesurer la lumière ambiante pour une station météo ;
- détecter la lumière dans une pièce ;
- suivre la lumière dans un robot ;
- détecter un passage ;
- ...



Ses symboles électriques 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.

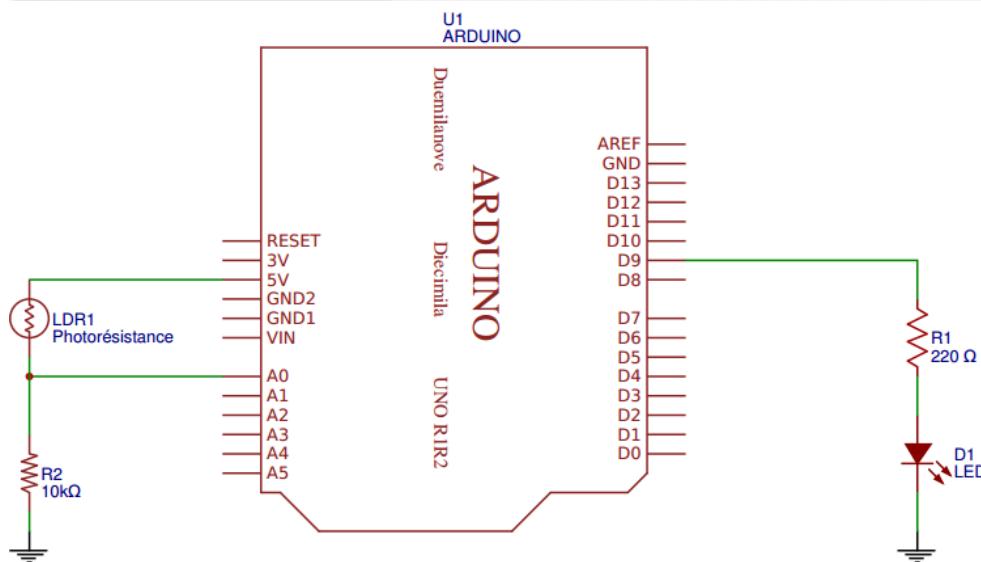
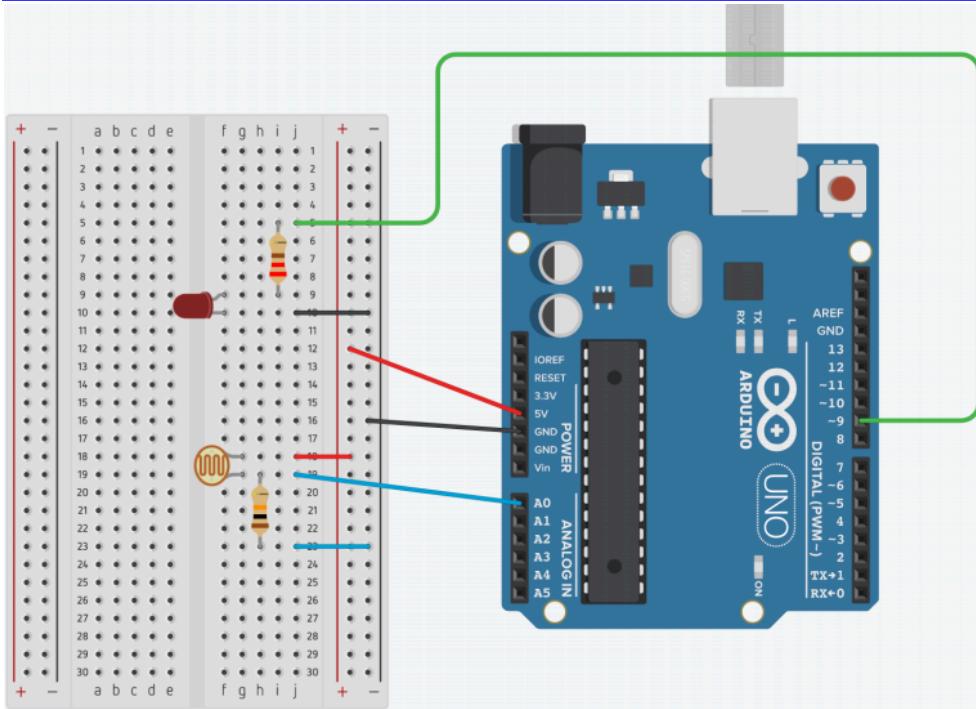
XVII-B - Circuit 11 : diviseur de tension

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**. 5 V 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 $220\ \Omega$;
- 1 résistance de $10\ k\Omega$;
- 1 photorésistance.

Circuit 11

XVII-C - 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 : valeur de seuil

```

1. /*
2.   Code 16 - Edurobot.ch, destiné à l'Arduino
3.   Objectif: Éteindre une LED dès que la luminosité est suffisante
4. */
5.
6. //***** EN-TÊTE DÉCLARATIF *****
7.
8. int lightPin = 0; //On renomme la broche A0 en "lightPin"
9. int ledPin = 9; //On renomme la broche 9 en "ledPin"
10.
11. //***** FONCTION SETUP = Code d'initialisation *****
12. // La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme
13. void setup()
14. {
15.     pinMode (ledPin, OUTPUT);
16.     Serial.begin(9600);
17. }
18.
19. void loop()
20. {
21.     int seuil = 900; //On définit un seuil de luminosité (sur 1023) à partir duquel la LED
22.     // s'éteint
23.     if (analogRead (lightPin)> seuil) //Si la luminosité est plus élevée que le seuil...
24.     {
25.         digitalWrite (ledPin, LOW); //... alors on éteint la LED.
26.     }
27.     else //Sinon...
28.     {
29.         digitalWrite (ledPin, HIGH); //...on allume la LED
30.     }
}

```

XVII-D - 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 fait 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 256×4 . 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

```

1. /*
2.   Code 17 - Edurobot.ch, destiné à l'Arduino
3.   Objectif: Faire varier la luminosité de la LED en fonction de la luminosité de la pièce
4.   Source: http://learn.adafruit.com/lights
5. */
6.
7. int lightPin = 0;
8.
9. int lightReading;
10.
11. int ledPin = 9;
12. int ledBrightness;
13.
14. void setup() {
15.     Serial.begin(9600);
16. }
17.
18. void loop() {

```

Code 17

```

19.   lightReading = analogRead(lightPin);
20.
21.
22.   Serial.print("Lecture analogique = ");
23.   Serial.println(lightReading);
24.
25.
26.   lightReading = 1023 - lightReading;
27.
28.   ledBrightness = lightReading / 4;
29.   analogWrite(ledPin, ledBrightness);
30.
31.   delay(100);
32. }
```

Analyse du code

Premièrement, on va définir nos constantes et nos variables. Ainsi, la broche A0, sur laquelle la photorésistance (couplée à la résistance *pull-down*) est connectée est renommée `lightPin`. On crée 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`. À noter que la broche 9 est compatible PWM. Enfin, on crée une variable `ledBrightness` pour y stocker la valeur de luminosité de la LED (comprise entre 0 et 255).

```

7. int lightPin = 0;
8. int lightReading;
9. int ledPin = 9;
10. int ledBrightness;
```

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

```
19. 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.

```

22.   Serial.print("Lecture analogique = ");
23.   Serial.println(lightReading);
24.   // Plus il fait sombre, plus la LED doit être brillante.
25.   // Cela signifie qu'il faut inverser la valeur lue de 0-1023 à 1023-0, avec une
   subtraction.
26.   lightReading = 1023 - lightReading;
27.
28.   //Enfin il faut faire correspondre les valeurs lues de 0-1023 à 0-255 qui sont les
   valeurs utilisées par analogWrite.
29.   // Pour cela, on convertit la valeur de la variable "lightReading" en la divisant par 4.
30.   ledBrightness = lightReading / 4;
31.   analogWrite(ledPin, ledBrightness); // On allume enfin la LED avec la luminosité stockée
   dans la variable "ledBrightness".
32.
33.   delay(100);
34. }
```

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

```
26. lightReading = 1023 - lightReading;
```

XVII-E - Code 18 : mappage de données

Le mappage des données est l'association des données appartenant à un ensemble avec les données appartenant à un autre ensemble, de manière que l'on puisse passer harmonieusement des premières aux secondes (11). 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

```

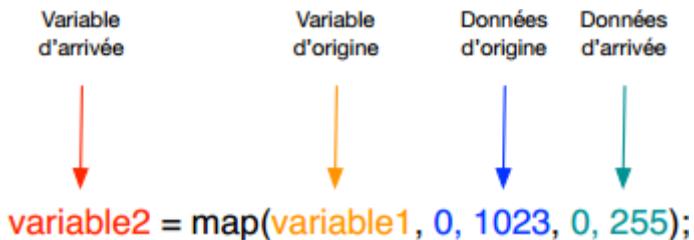
1. /*
2.     Code 18 - Edurobot.ch, destiné à l'Arduino
3.     Objectif : Faire varier la luminosité de la LED en fonction de la luminosité de la pièce
4.     avec mappage de données
5.     Source : http://learn.adafruit.com/lights
6.
7. int lightPin = 0; // La photorésistance et la résistance pulldown de 1 à 10K sont connectées
8.     à A0
9. int lightReading; // Lecture analogique de la valeur de la photorésistance avec le diviseur
10.    de tension
11. int ledPin = 9; // LED en PWM sur la broche 9
12. int ledBrightness; // Variable pour stocker la valeur de luminosité de la LED
13.
14. void setup() {
15.     Serial.begin(9600);
16.
17. void loop() {
18.     lightReading = analogRead(lightPin);
19.     // Affichage de la valeur de "lightReading" dans la console
20.     Serial.print("Lecture analogique = ");
21.     Serial.println(lightReading); // Affichage de la valeur brute issue de la
22.     photorésistance.
23.     // Plus il fait sombre, plus la LED doit être brillante.
24.     // Cela signifie qu'il faut inverser la valeur lue de 0-1023 à 1023-0, avec une
25.     soustraction.
26.     lightReading = 1023 - lightReading;
27.     // Enfin il faut faire correspondre avec la fonction "map" les valeurs lues de 0-1023 à
28.     // 0-255 qui sont les valeurs utilisées par analogWrite.
29.     ledBrightness = map(lightReading, 0, 1023, 0, 255);
30.     analogWrite(ledPin, ledBrightness); // On allume enfin la LED avec la luminosité stockée
31.     dans la variable "ledBrightness".
32. }

```

On voit à la ligne suivante le mappage :

```
28. 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.

XVIII - 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



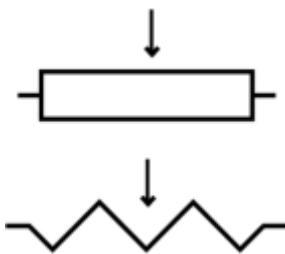
Potentiomètre linéaire

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



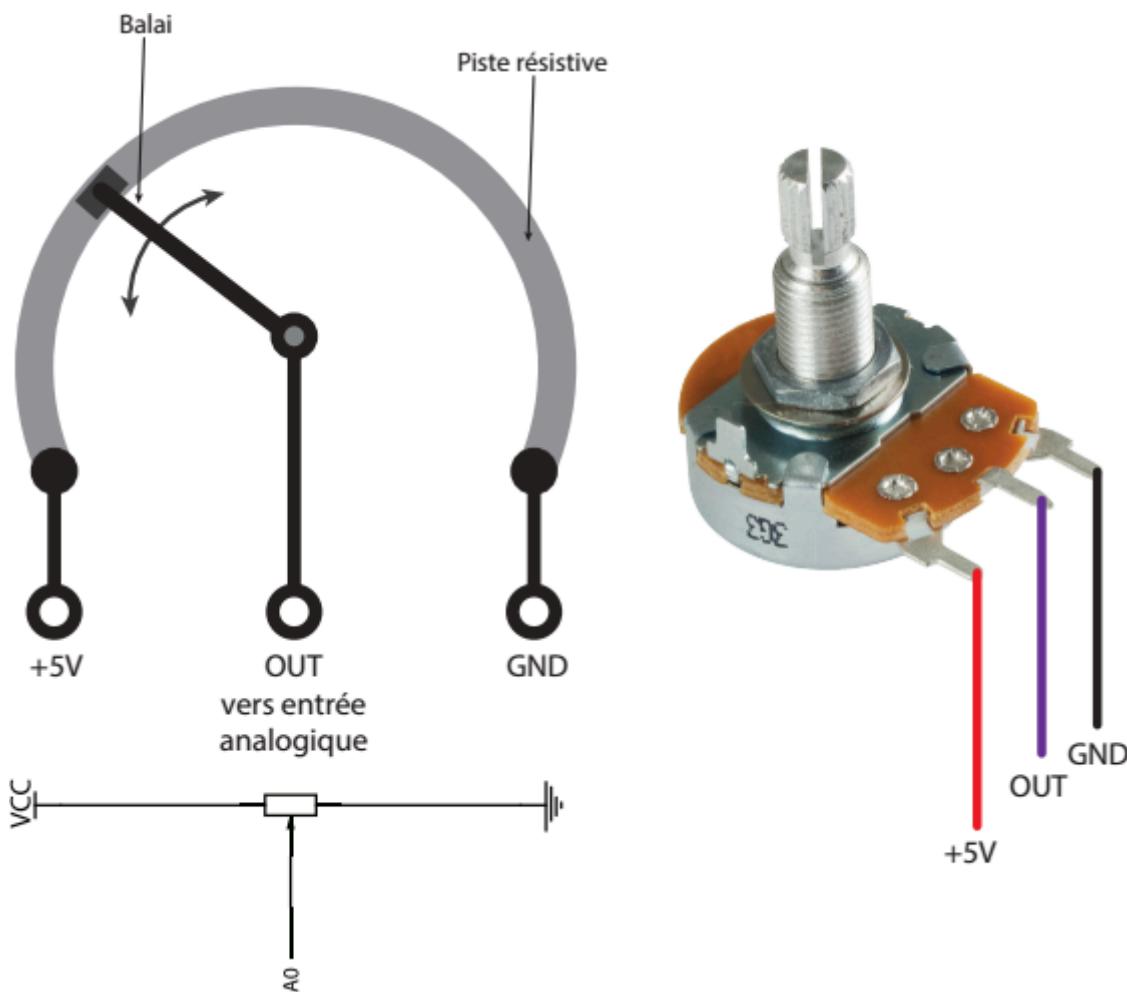
Table de mixage du RadioBus (<http://radiobus.fm>)

Voici les symboles électriques (européens dessus et américains 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.

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

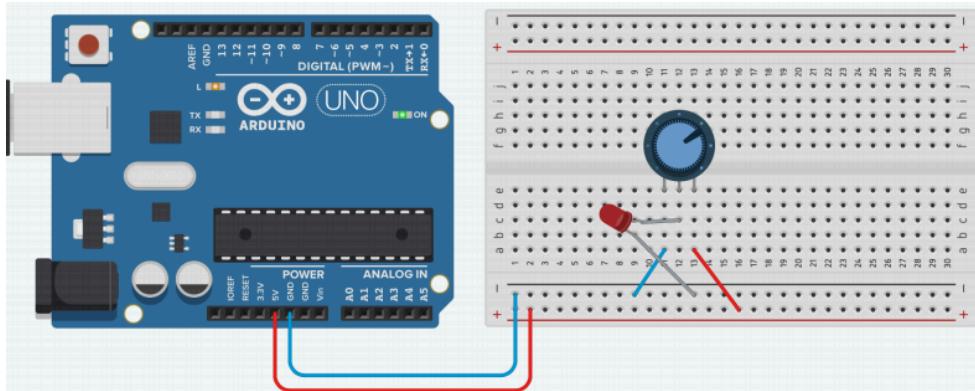


XVIII-A - Circuit 12 : utiliser le potentiomètre pour faire varier la luminosité d'une LED

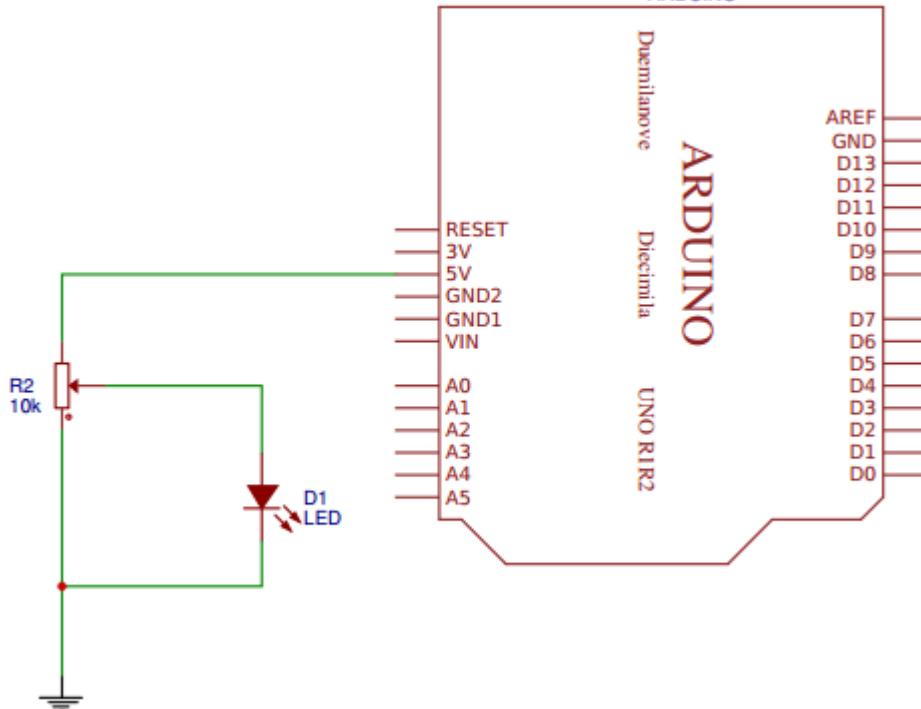
Il est temps de griller notre première LED (si cela n'est pas déjà fait) ! C'est le passage obligé de tout bon *maker* ! Pour ce faire, nous allons connecter une LED sur la broche *OUT* du potentiomètre et la mettre à la terre. Petit conseil : avant de brancher, tourner le potentiomètre au minimum, dans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



U1 ARDUINO



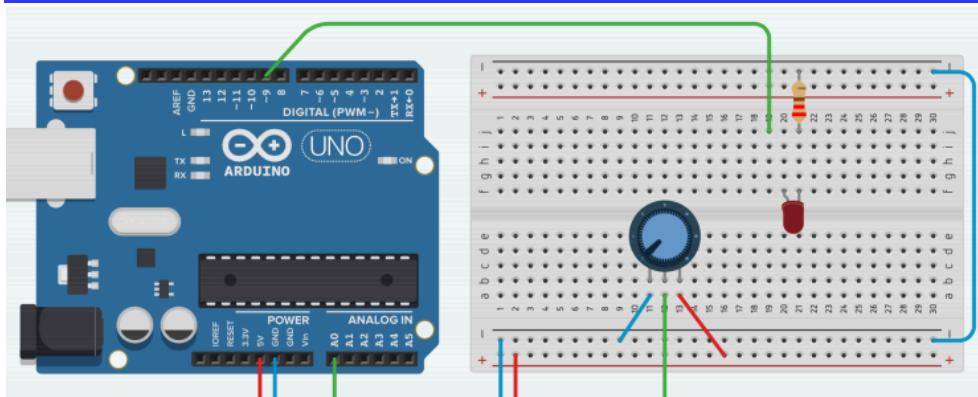
Nous venons de le voir : varier le voltage à l'entrée d'une LED n'est pas la meilleure solution pour faire varier sa luminosité... même si la LED ne grille pas, au-delà de son voltage de fonctionnement, sa durée de vie sera fortement réduite. Mais qu'importe, puisque nous avons la fonction PWM (12) . Nous allons donc coupler le potentiomètre, en entrée analogique sur A0, avec une LED en PWM sur la broche 9.

Liste des composants

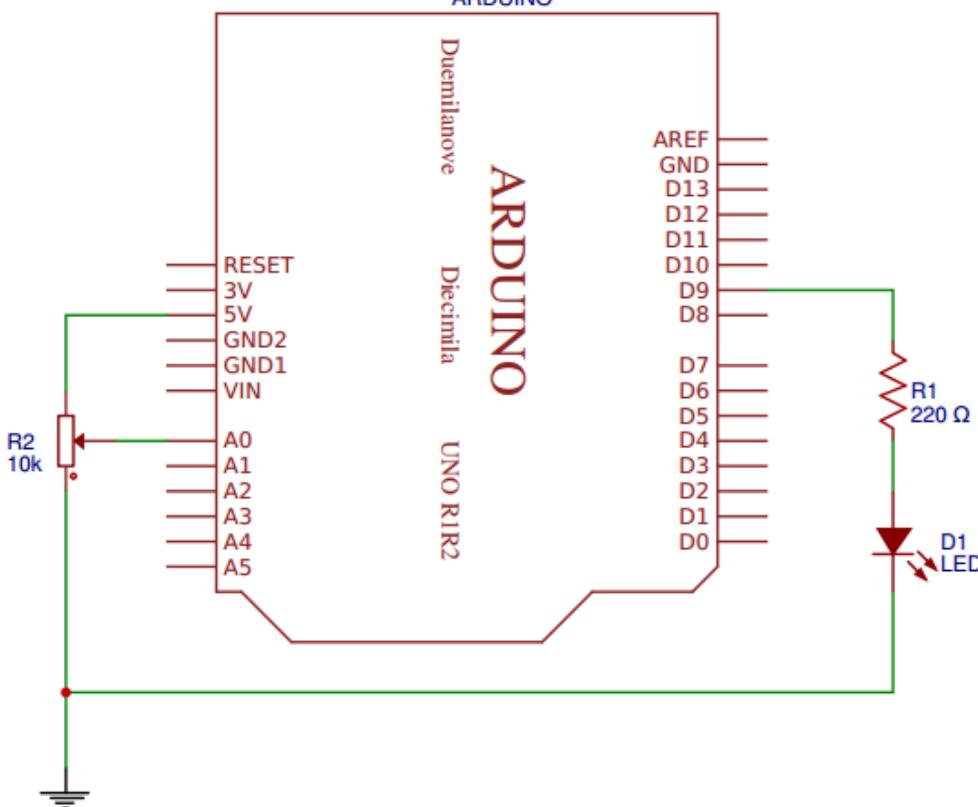
- 1 LED ;
 - 1 résistance de $220\ \Omega$;
 - 1 potentiomètre de $10\ k\Omega$ ou $20\ k\Omega$.

XVIII-B - Circuit 13 : utiliser le potentiomètre pour faire varier la luminosité d'une LED

Circuit 13



U1
ARDUINO



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

```

1. /*
2.   Code 19 - Edurobot.ch, destiné à l'Arduino
3.   Objectif: Lire la valeur du potentiomètre pour faire varier la luminosité de la LED en PWM
4.   Source: Philippe Krähenbühl
5. */
6.
7. int ledPin = 9;      // On renomme la broche 9 en "ledPin"
8. int analogPin = 0;  // Le potentiomètre est connecté à la broche analogue 0
9. int val = 0;        // la variable qui stocke la valeur lire est mise à 0

```

Code 19

```
10.  
11. void setup()  
12. {  
13.     pinMode(ledPin, OUTPUT);  
14. }  
15.  
16. void loop()  
17. {  
18.     val = analogRead(analogPin); //lecture de la valeur analogue (potentiomètre)  
19.     analogWrite(ledPin, val / 4); // valeur analogue lue (0 à 1023) divisée par 4 (0 à 255)  
20. }
```

XVIII-C - Circuit 14 : allumer les LED d'un bargraphe à l'aide d'un potentiomètre

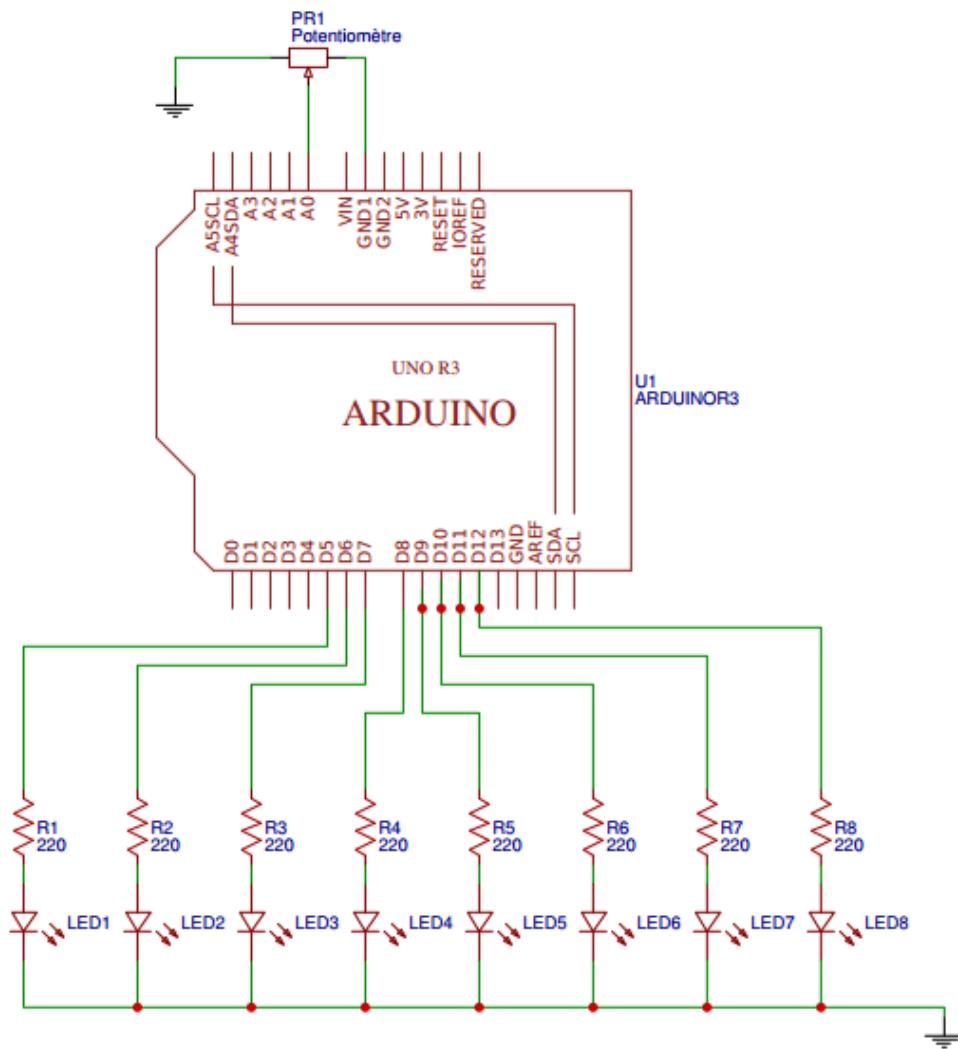
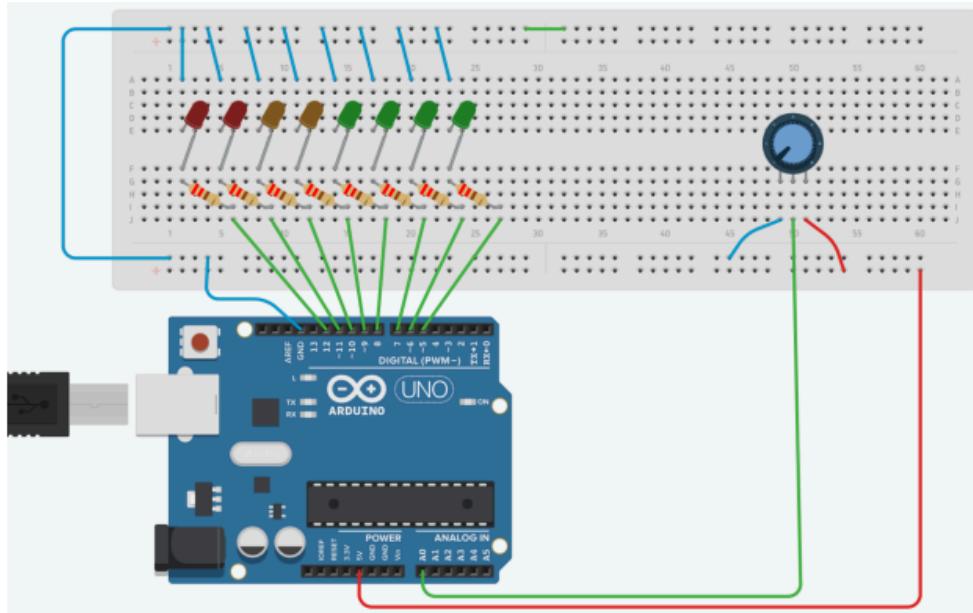
Quand on tourne le bouton pour augmenter le volume, il serait intéressant d'avoir un bargraphe qui nous indique le niveau.

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

Liste des composants

- 8 LED ;
- 8 résistances de 220 Ω ;
- 1 potentiomètre de 10 kΩ ou 20 kΩ.

Circuit 14



XVIII-D - 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 LED devront être allumées. Ainsi, lorsqu'il recevra la valeur de 512, 4 LED 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 LED 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
 LED8 : de 769 à 1023

À 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 (voir [code 17](#)).

Code 20

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

// Information sur les LED

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() {
    /*Écriture de la valeur du potentiomètre
     dans la variable "readValue"
    */
    readValue = analogRead(potPin);
```

Code 20

```

/*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);
}

```

Code 20

```

        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);
    }

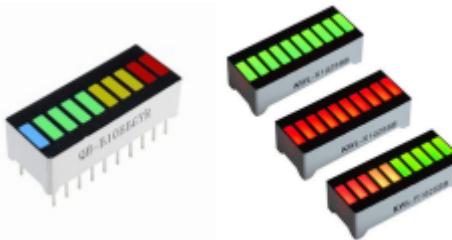
    // De 768 à 895, led 7
    else if (readValue >= 768 && readValue < 896) {
        Serial.println("\tled 7");
        digitalWrite(led1, HIGH);
        digitalWrite(led2, HIGH);
        digitalWrite(led3, HIGH);
        digitalWrite(led4, HIGH);
        digitalWrite(led5, HIGH);
        digitalWrite(led6,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

```

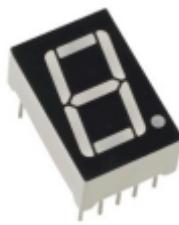
Variante 1 : utiliser un bargraphe 10 LED

On peut, à ce stade, utiliser un potentiomètre et afficher sa valeur à l'aide d'un bargraphe 10 LED. 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).



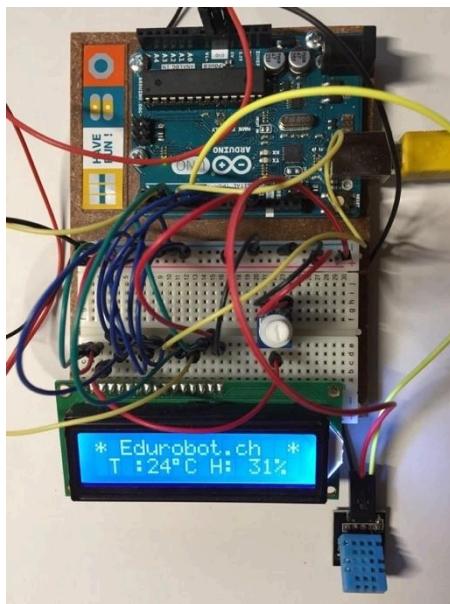
XIX - Projet 11 : construire une station météo

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

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

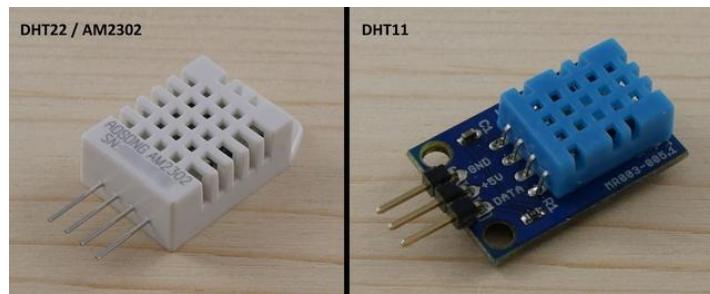
Connaître la température et l'humidité est une information utile pour vivre confortablement, pour stocker des aliments ou du matériel, pour des prévisions météo... Avec ce projet nous apprendrons comment mesurer ces deux données et à les afficher sur un écran LCD.

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.



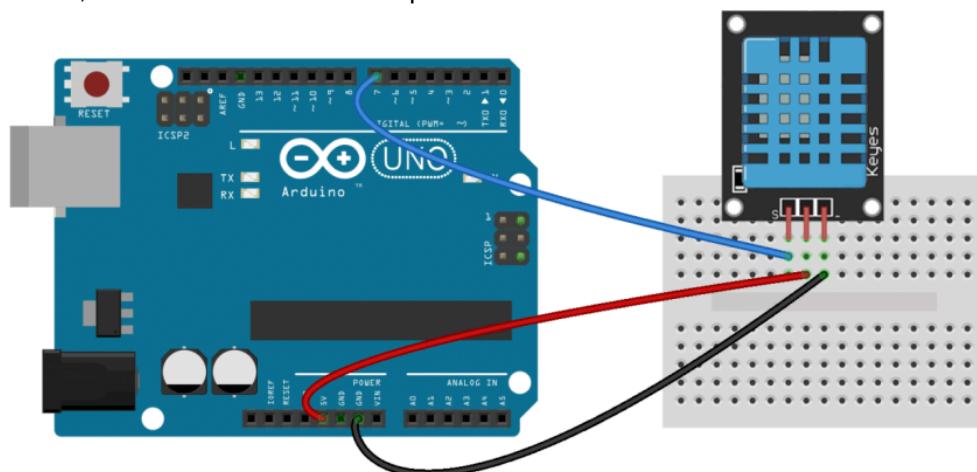
Attention : il existe des versions des capteurs DHTxx qui ne sont pas prémontés sur un petit circuit comme le modèle de gauche ci-dessus utilisé dans ce projet. Si c'est le cas, il faut ajouter une résistance de tirage de 5 à 10 kOhms et un condensateur de 100 nF entre les broches 1 (alimentation 5 V) 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 **5 V** 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.

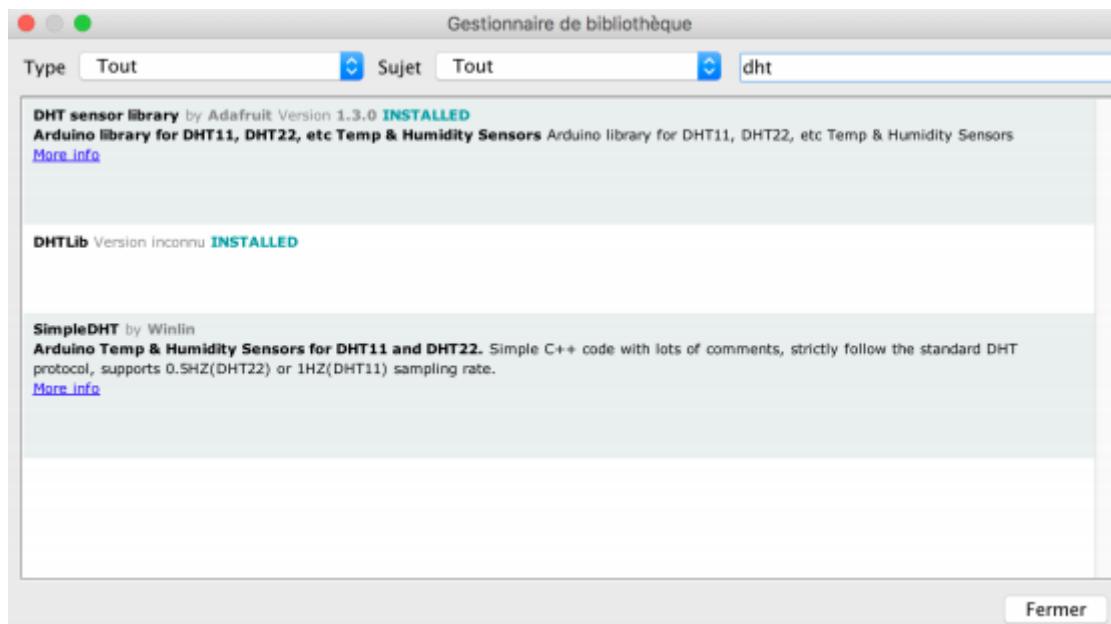
Une fois ces données connues, le montage est un jeu d'enfant et un schéma est ici inutile, nous allons choisir le Pin7 pour les données :



XIX-A - 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 autre solution 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

```

1. /*
2.   Code 21 - Edurobot.ch, destiné à l'Arduino
3.   Objectif: Afficher la température et l'humidité sur un écran
4. */
5.
6. //***** EN-TÊTE DÉCLARATIF *****
7. #include <dht.h> // on charge la bibliothèque
8.
9. dht DHT;           // on crée l'objet du capteur DHT11
10. #define DHT11_PIN 7 // on définit le Pin qui sera utilisé pour recevoir les données
11.
12. void setup() // début de la fonction setup()
13. {
14.     Serial.begin(9600); // Pour une fois, l'ouverture du port série et la définition de
15.                         // sa vitesse en bauds est utile, même indispensable puisque l'on
16.                         // va utiliser ce port pour afficher les valeurs du capteur.
17. }
18.

```

Code 21

```

19. //***** FONCTION LOOP = Code d'initialisation *****
20. // La fonction setup() est exécutée en premier et une seule fois, au démarrage du programme
21.
22. void loop() // début de la fonction loop()
23. {
24.     int chk = DHT.read11(DHT11_PIN); // on lit l'état du capteur
25.     Serial.print("Temperature = ");
26.     Serial.println(DHT.temperature); // on affiche sur la même ligne la température lue
27.     Serial.print("Humidite = ");
28.     Serial.println(DHT.humidity); // on affiche sur la même ligne l'humidité lue
29.     delay(1000); // on fait une pause de 1 seconde entre 2 interrogations du capteur
30. }

```

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

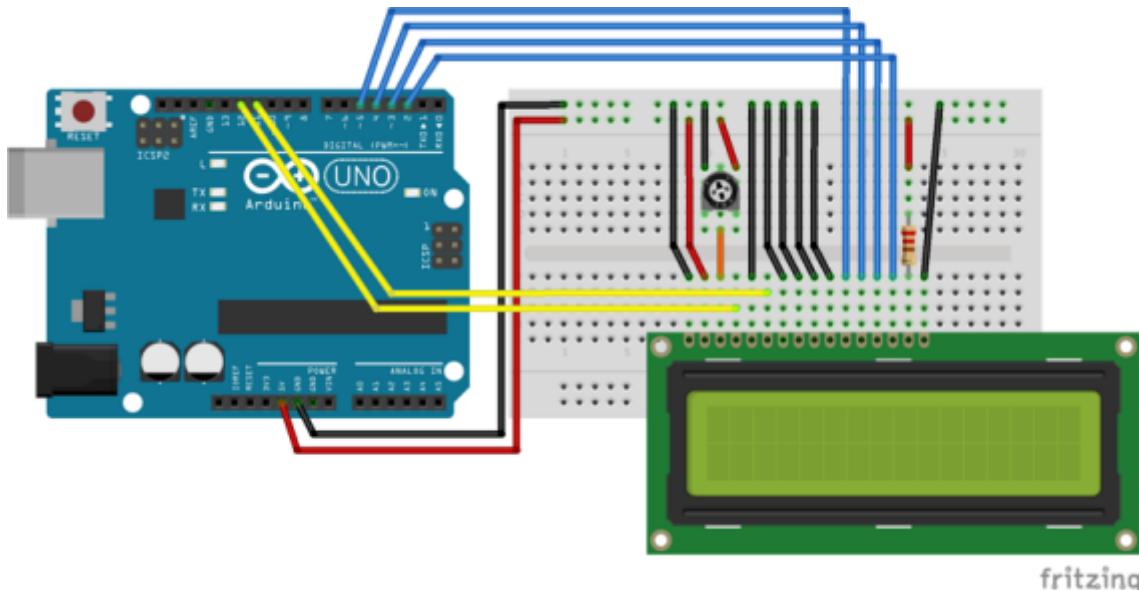
Afficher le résultat des mesures sur le moniteur série est bien sympathique, mais si l'on veut faire une installation de mesure permanente et autonome, cette solution ne convient pas, car il faut que l'ordinateur soit en permanence allumé et branché à la carte. L'affichage sur un petit écran LCD est une solution bien plus élégante et pratique. L'un des écrans LCD les plus répandus 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). À 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.

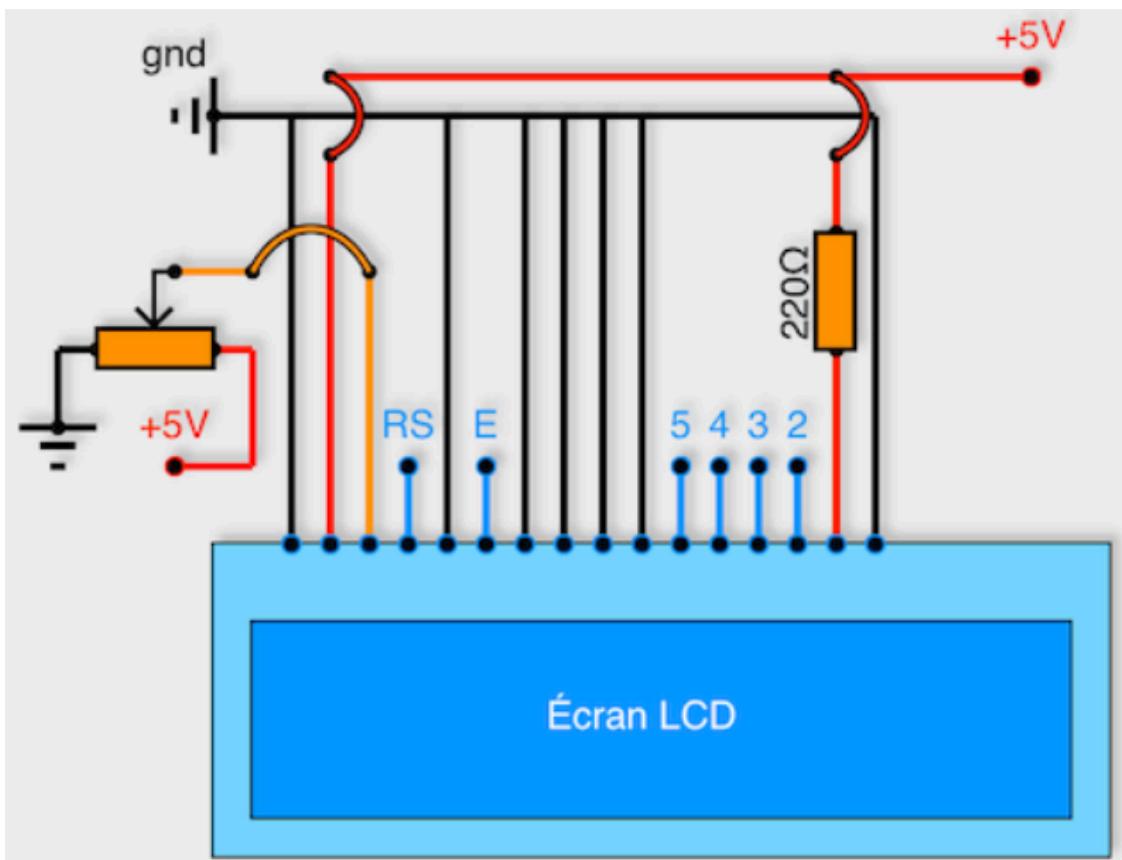
Liste des composants :

- 1 écran LCD 2x16 ;
- 1 potentiomètre (résistance variable) ;
- 1 résistance de $220\ \Omega$;
- 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 mâle-mâle 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 (5 V) pour le 2^e
- Le 3^epin est connecté à un potentiomètre et sert à régler le contraste de l'écran LCD
- Le 4^e, 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 5^e doit être connecté au *ground* (GND)
- Le 6^e, noté E pour *Enable*, est connecté au *pin* 11 de l'Arduino. Il permet de lancer ou non l'écriture dans les zones mémoire
- Les quatre suivants (7, 8, 9 et 10^e) sont reliés au *ground* (GND)
- Les quatre qui suivent (11^e à 14^e, 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 16^e) servent pour alimenter la LED du rétroéclairage de l'écran LCD. Attention, l'avant-dernier (pôle positif 5 V) 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.

XIX-B - Code 22 : afficher les données sur l'écran LCD

Contrairement à la bibliothèque « **DHTLib** » utilisée auparavant, la bibliothèque « **LiquidCrystal** » est 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

```

1. /*
2.   Code 22 - Edurobot.ch, destiné à l'Arduino
3.   Objectif: Afficher la température et l 'humidité sur un écran LCD
4. */
5.
6. //***** EN-TÊTE DÉCLARATIF *****
7. #include <dht.h>           // on inclut la bibliothèque pour le capteur DHT11
8. #include <LiquidCrystal.h>  // on inclut la bibliothèque pour l'écran LCD
9.
10. LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // on crée l'objet LCD et on définit les Pins utilisés
11.
12. dht DHT;                  // on crée l'objet du capteur DHT11
13.
14. #define DHT11_PIN 7        // on définit le Pin utilisé pour les données du DHT11
15.
16. void setup(){
17.   lcd.begin(16, 2);      // on initialise la communication avec l'écran LCD
18. }
19. void loop()
20. {
21.   int chk = DHT.read11(DHT11_PIN); // on lit les données du capteur DHT
22.   lcd.setCursor(0,0);          // on place le curseur de l'écran LCD au début de la
  1ère ligne
23.   lcd.print("Temp: ");       // on écrit le mot "Temp: " à l'emplacement du curseur
24.   lcd.print(DHT.temperature,1); // on écrit la température lue par le capteur, avec 1
  chiffre derrière la virgule
25.   lcd.print((char)223);      // on ajoute le symbole ° après la valeur de la
  température
26.   lcd.print("C");           // on ajoute la lettre C pour degré Celsius
27.   lcd.setCursor(0,1);       // on déplace le curseur de l'écran LCD au début de la
  2eline
28.   lcd.print("Humidity: ");  // on écrit le mot "Hum. rel: " à l'emplacement du
  curseur
29.   lcd.print(DHT.humidity,1); // on écrit l'humidité relative lue par le capteur, avec
  1 chiffre derrière la virgule
30.   lcd.print("%");          // on ajoute le symbole "%" après la valeur de
  l'humidité
31.   delay(1000);            // on attend une seconde avant de procéder à la lecture
  suivante
32. }
```

XX - Projet 12 : utiliser un servomoteur

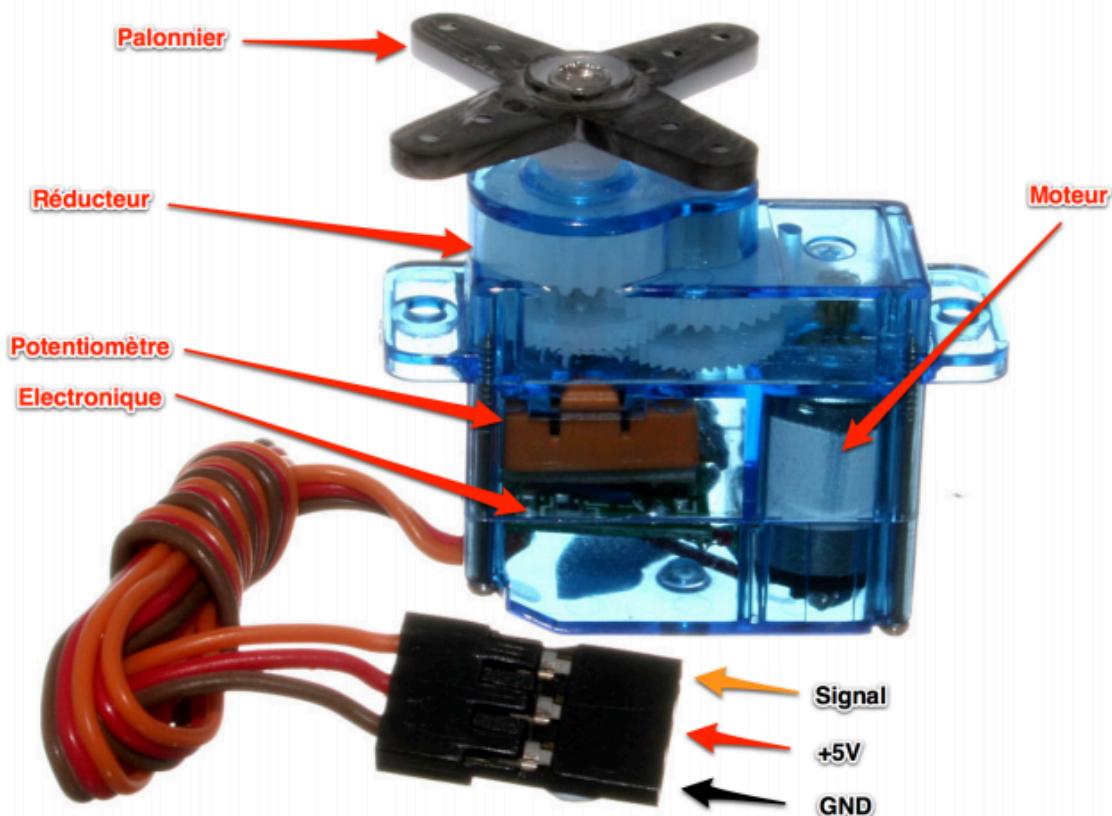
Les servomoteurs, souvent abrégés en « servo » tout court par leurs utilisateurs, sont des moteurs d'un type particulier, très appréciés pour faire tourner quelque chose jusqu'à une position bien précise et capable de maintenir cette position jusqu'à l'arrivée d'une nouvelle instruction. Ils sont très 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 9 g, 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 sur 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 rotation continue, 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 :



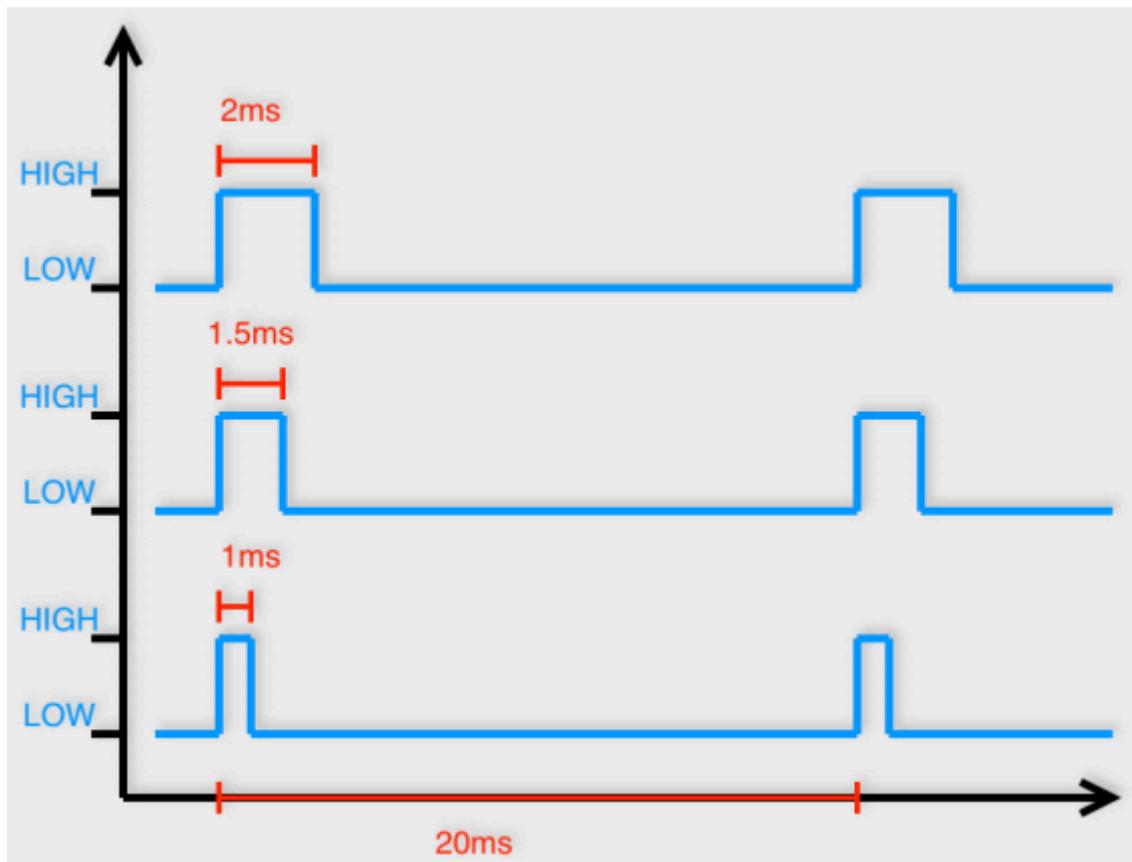
Un petit moteur à courant continu est relié à un potentiomètre (résistance variable) par l'intermédiaire d'un circuit électronique ce qui permet de contrôler finement le moteur en fonction de la position du potentiomètre. Sur l'axe de sortie du moteur, une série d'engrenages permet d'augmenter son couple (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 finalement déterminera la position du bras n'est pas la période, mais bien la durée de l'impulsion :

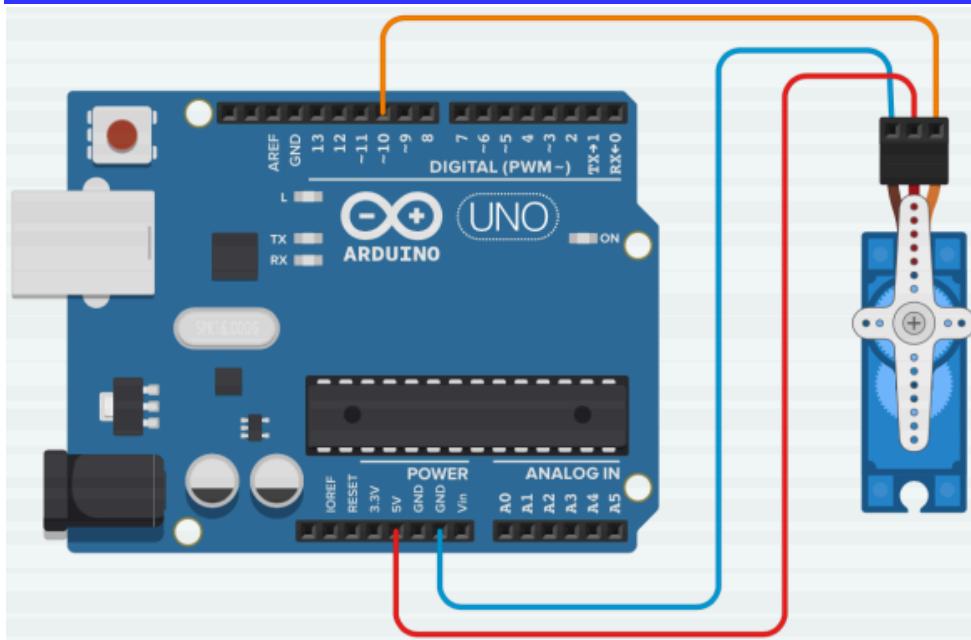
- pour une impulsion de 1 ms, le servomoteur se met en position 0° ;
- pour une impulsion de 1,5 ms, le servomoteur se met en position 90° ;
- pour une impulsion de 2 ms, le servomoteur se met en position 180° .



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 (5 V), 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



XX-A - 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 trois 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

```

1. /*
2.   Code 23 - Edurobot.ch, destiné à l'Arduino
3.   Objectif : Faire bouger le bras d'un servomoteur dans un sens puis dans l'autre,
4.   indéfiniment
5.
6. //*****EN-TÊTE DECLARATIF*****
7. #include <Servo.h> // on inclut la bibliothèque pour piloter un servomoteur
8.
9. Servo monServo; // on crée l'objet monServo
10.
11. void setup()
12. {
13.   monServo.attach(9); // on définit le Pin utilisé par le servomoteur
14. }
15.
16. void loop()
17. {
18.   for (int position = 0; position <=180; position ++){ // on crée une variable position qui
19.     prend des valeurs entre 0 à 180 degrés
20.     monServo.write(position); // le bras du servomoteur prend la position de la variable
21.     position
22.     delay(15); // on attend 15 millisecondes
23.   }
24.   for (int position = 180; position >=0; position --){ // cette fois la variable position
25.     passe de 180 à 0°
26.     monServo.write(position); // le bras du servomoteur prend la position de la variable
27.     position
27.   }

```

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 incrémentations utilisées, 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 maintenant 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 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.

XX-B - Code 24 : servomoteur et gestion des tâches

Code 24

```

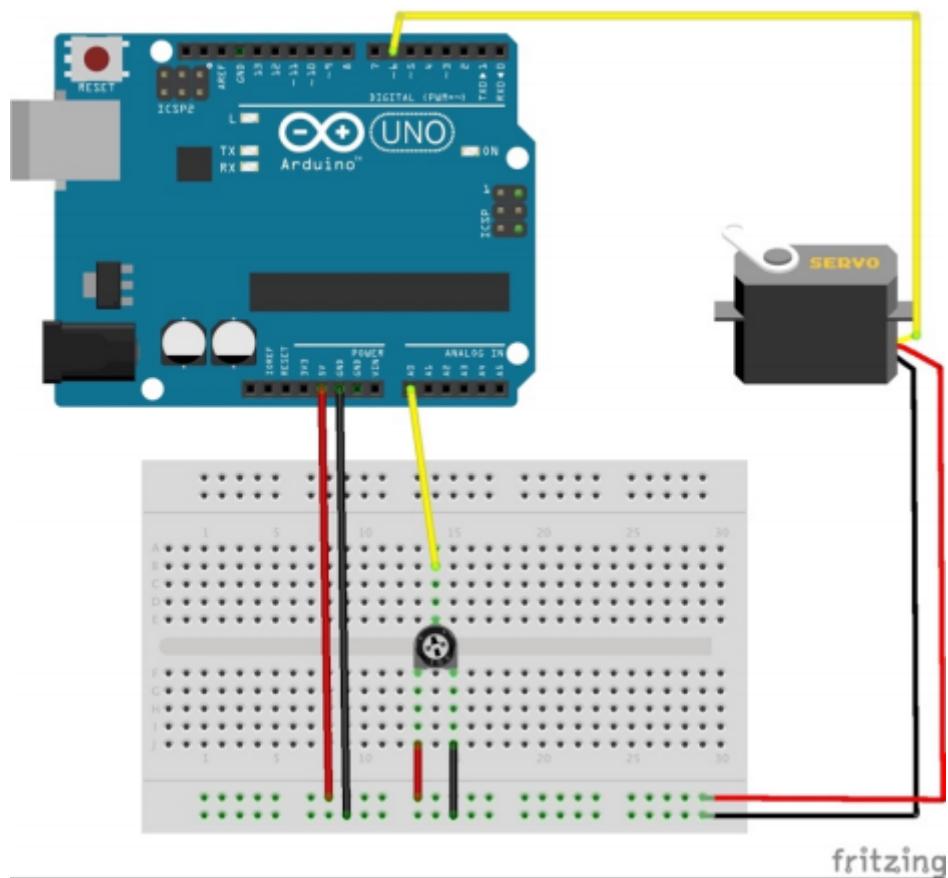
1. /*
2.  Code 24 - Edurobot.ch, destiné à l'Arduino
3.  Objectif : prouver que la bibliothèque Servo permet au servomoteur d'agir et de se maintenir
4.          en position même lorsque l'Arduino effectue une autre tâche.
5. */
6.
7. //*****EN-TÊTE DECLARATIF*****
8. #include <Servo.h> // on inclut la bibliothèque pour piloter un servomoteur
9.
10. Servo monServo; // on crée l'objet monServo
11.
12. void setup()
13. {
14.     monServo.attach(9); // on définit le Pin utilisé par le servomoteur
15.     pinMode(13,OUTPUT); // la Pin13 est mise en mode OUTPUT
16. }
17.
18. void loop()
19. {
20.     monServo.write(0); // on dit à l'objet de mettre le servo à 0°
21.     diode13(); // appel de la fonction diode13 qui est définie plus bas
22.     monServo.write(180); // on dit à l'objet de mettre le servo à 180°
23.     diode13(); // appel de la fonction diode13
24. }
25.
26. void diode13() //on va faire clignoter 15 fois la diode 13
27. {
28.     for (int t=0;t<15;t++){
29.         digitalWrite(13,HIGH);
30.         delay(100);
31.         digitalWrite(13,LOW);
32.         delay(100);
33.     }
34. }

```

À 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 tournez qui servira à positionner le bras du servomoteur.

Pour vous aider, voici le schéma du montage :



fritzing

XX-C - Code 25 : commander un servomoteur avec un potentiomètre

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

Code 25

```

1. /*
2.   Code 25 - Edurobot.ch, destiné à l'Arduino
3.   Objectif : commander un servomoteur avec un potentiomètre
4. */
5.
6. //*****EN-TÊTE DECLARATIF*****
7. #include <Servo.h> // on inclut la bibliothèque pour piloter un servomoteur
8.
9. Servo monServo; // on crée l'objet monServo
10.
11. int pinmonServo = 9; // on définit la Pin9 liée à la commande du servomoteur
12. int pinPotar = A0; // on définit la Pin analogique A0 pour la lecture du potentiomètre
13.
14. void setup()
15. {
16.   monServo.attach(pinmonServo); // on lie l'objet monServo au pin de commande
17. }
18.
19. void loop()
20. {
21.   int valeurPotar=analogRead(pinPotar); // on lit la valeur du potentiomètre
22.   int angle=map(valeurPotar, 0,1023,0,180); // on transforme la valeur analogique lue en
      valeur d'angle entre 0 et 180°
23.   monServo.write(angle); // on met le bras du servomoteur à la position
      angle
24. }
```

1 : <http://www.edurobot.ch/?p=1251>

2 : Histoire de l'Arduino : <http://www.framablog.org/index.php/post/2011/12/10/arduino-histoire>

3 : Références : <http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino> et

4 : Ce sera la solution utilisée principalement dans ce cours. Pour plus d'informations : <http://mitic.education/?p=1878>.

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

6 : Source : http://mediawiki.e-apprendre.net/index.php/D%C3%A9couverte_des_microcontr%C3%B4leurs_avec_le_Diduino

7 : Mais on aurait aussi pu l'appeler petite_fleur...

8 : Disponible ici : www.scolcast.ch/episode/arduino-lecole-apprendre-compter

9 : Source de la police : <http://www.dafont.com/7led.font>

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

11 : <https://fr.wiktionary.org/wiki/mappage> et <https://www.arduino.cc/reference/en/language/functions/math/map/>

12 : Voir [Projet 7](#) de ce cours.