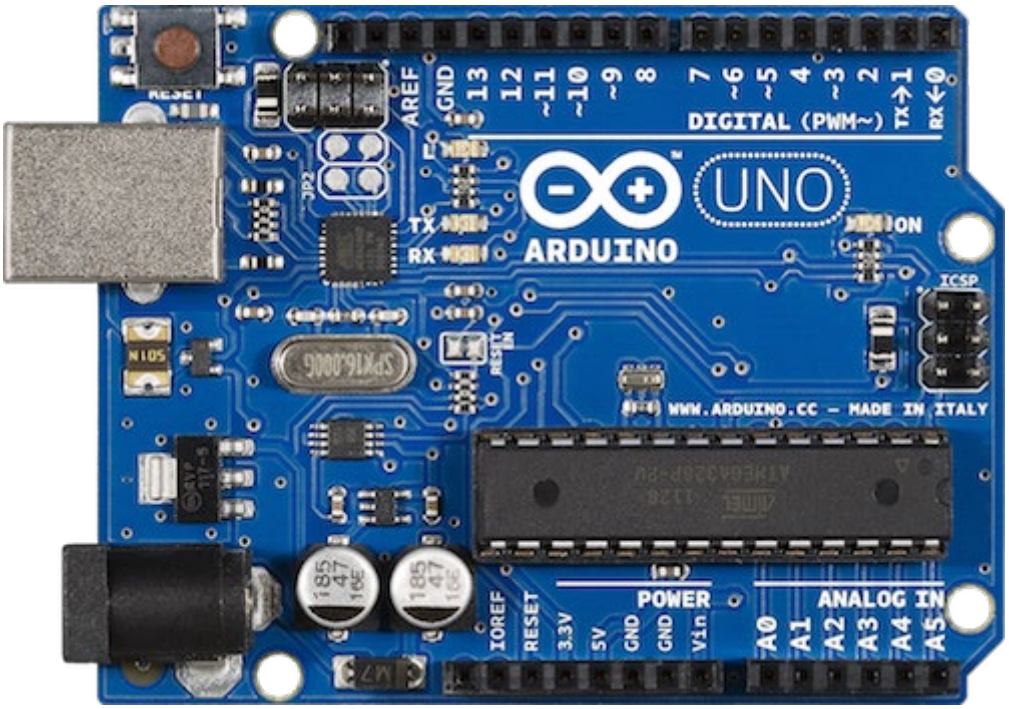
PARTIE 1 : Les sorties

Réaliser vos premiers montages avec Arduino

Les objets connectés, ça vous branche ? Vous rêvez de construire un robot autonome, de contrôler votre maison depuis votre mobile, ou encore de réaliser votre propre guirlande de Noël ? Ce cours est fait pour vous ! Il va vous permettre d'apprendre à programmer une carte Arduino pour connecter des composants et réaliser toutes sortes de projets allant de micro-robotique à art numérique, en passant par domotique, électricité ou encore mécanique.

Voici à quoi ressemble la carte Arduino :



La carte Arduino Uno

Et voici quelques exemples de ce que vous pouvez réaliser avec une telle carte :

- Un robot mobile capable d'éviter les obstacles ou de suivre une ligne au sol ;
- Une interface entre votre téléphone mobile et les éclairages de votre maison ;
- Des afficheurs d'informations à base de textes défilants sur des panneaux à LEDs ;
- Une station météorologique consultable sur le Web ;
- Un pilote de caméra de surveillance.

De plus en plus utilisé dans le monde de l'éducation, l'Arduino est aussi apprécié dans le cadre des technologies embarquées, du modélisme et peut rapidement devenir un hobby. La force de cette carte Arduino à microcontrôleur est, au-delà de son prix, la facilité avec laquelle on peut réaliser des petits projets viables, et surtout la communauté libre qui s'est développée autour d'elle.

L'objectif de ce cours est donc de vous initier à la programmation et la réalisation de montages avec une carte Arduino.

Table des matières

Objectifs pédagogiques :

À la fin de ce cours, vous saurez...

- Programmer une carte Arduino avec le logiciel associé ;
- Contrôler le clignotement de LED dans les montages
- Utiliser les éléments de programmation de base pour l'Arduino
- Utiliser des servo-moteurs, moteurs à courant continu
- Créer des montages interactifs grâce aux capteurs électroniques

Pré-requis : Aucun

Mais, pour profiter pleinement du cours, vous aurez besoin d'un kit Arduino dit "kit du débutant", comprenant une carte Arduino UNO R3, une breadboard, des fils de connexion et quelques composants électroniques. Pas de panique, je reviens sur ce matériel en début de cours !

1. Partie 1 – Les sorties

1. Installez vos outils de travail

Comme pour toute chose, il faut commencer par le commencement. Le monde de l'Arduino est immense. On peut s'y perdre (toujours avec plaisir), on peut s'y disperser et passer à côté de l'essentiel. C'est un voyage à préparer. Il faut savoir où l'on va, comment on y va, ce qu'on doit prendre et connaître, les risques, les attentes, les plaisirs, les déceptions, la durée et tant d'autres détails qui feront de cette expérience une aventure unique et addictive. Quel lyrisme ! Bref tout ça pour dire qu'en lisant cette partie, vous saurez quelle est la voie que je vous propose de suivre jeunes padawans : découvrir ce qu'est l'Arduino, à qui il peut convenir, comment se le procurer, quels outils installer et quel matériel acquérir.

À qui s'adresse ce cours ?

La programmation sur Arduino n'est pas que de la programmation. C'est d'ailleurs ce qui fait son intérêt. Si l'on s'engage assez loin, on touche finalement à tout ce qui concerne les branches de la robotique et de la domotique : la programmation, l'électronique, la mécanique, l'interfaçage, les réseaux, la programmation graphique, la FAO... Attirant et effrayant à la fois !

On peut dire que ce cours est fait pour vous si :

- Vous êtes novice en électronique et en informatique et ces domaines vous intéressent ou vous passionnent (c'est encore mieux) et vous voulez passer à l'acte ;
- Vous êtes développeur, et l'interfaçage électronique vous intéresse, ou inversement, vous êtes électronicien et l'envie de développer vous prend ;
- Vous êtes du type bricolo-électro-touche-à-tout et vous débordez de projets ;
- Vous êtes enseignant et vous souhaitez vous lancer dans la micro-robotique avec vos élèves ;
- Vous êtes un adolescent qui erre entre son lit et le frigo, mais dans le fond vous savez que vous valez mieux que ça, et il n'y a pas de raison qu'Anakin Skywalker ait pu fabriqué Z6PO tout seul et pas vous, force ou pas force.

La liste est longue finalement, car l'Arduino permet de réaliser des applications dans des domaines tellement variés que je n'en citerai que quelques uns : éducation, photographie, arts numériques, micro-robotique, astronomique, technologie, domotique, sécurité...

Attention, tout ce qui peut être réalisé avec l'Arduino trouve son équivalent dans le commerce de façon plus compacte et souvent plus ergonomique. L'utilisation de l'Arduino reste en grande partie ludique et s'adresse à des projets pour des particuliers ou des étudiants. Bien qu'en s'appliquant certains obtiennent des résultats proches d'une conception professionnelle.

Vous ne construirez pas R2D2 avec un Arduino, mais un robot aspirateur est tout à fait réalisable ! Mais ce serait bien réducteur de résumer les potentialités de l'Arduino à cela.

Une contrainte en revanche est à connaître, qui peut arrêter certain(e)s dans leur élan : l'Arduino est payant, ainsi que beaucoup des composants dont vous aurez besoin. Il ne s'agit pas d'un simple téléchargement sur le Web, mais bien de matériel qui nécessite un léger investissement (environ 50€ pour bien débuter), vous trouverez les détails et des indications de tarifs dans les sections suivantes.

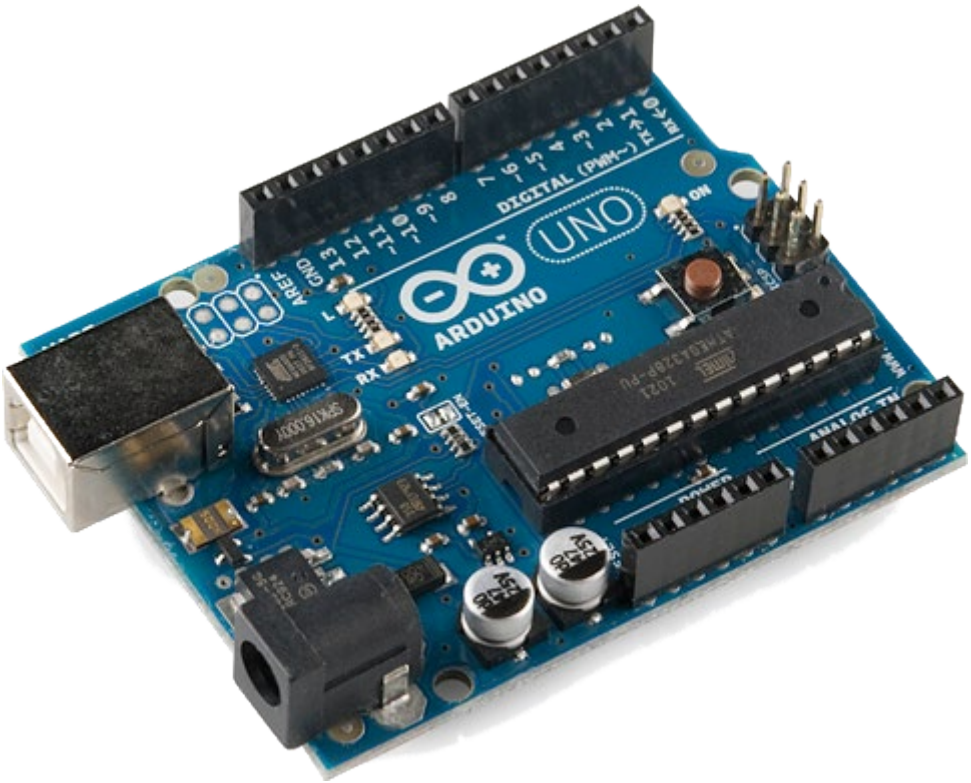
Si après ce préambule, vous êtes toujours partant pour apprendre et créer avec un Arduino, alors entrons dans le vif du sujet !

Qu'est-ce que l'Arduino ?

Bonne question, allons voir tout d'abord du côté de Wikipédia. J'ai allégé leur définition et j'ai mis en gras le vocabulaire à connaître en l'expliquant ensuite.

Arduino est un **circuit imprimé** en **matériel libre** sur lequel se trouve un **microcontrôleur** qui peut être programmé pour analyser et produire des **signaux électriques**.

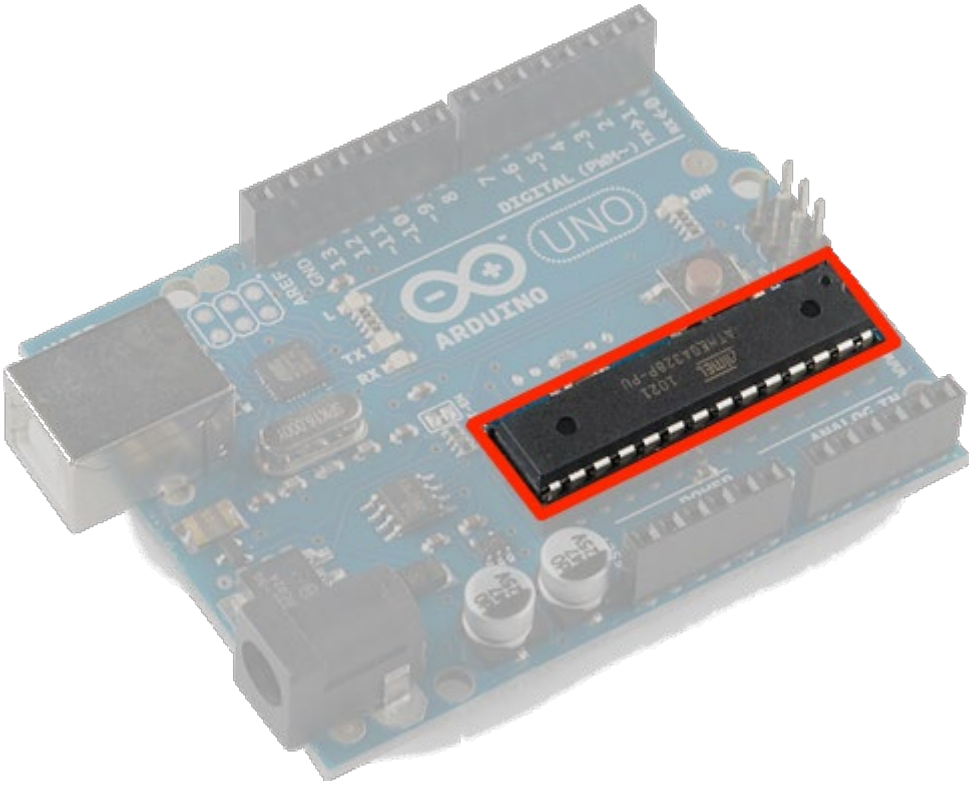
- **Circuit imprimé :** C'est une sorte de plaque sur laquelle sont soudés plusieurs composants électroniques reliés entre eux par un circuit électrique plus ou moins compliqué. L'Arduino est donc un circuit imprimé. La photo donne une idée de la taille par rapport à la connexion USB carrée (à gauche sur la photographie, la même que sur votre imprimante par exemple).



La carte Arduino UNO

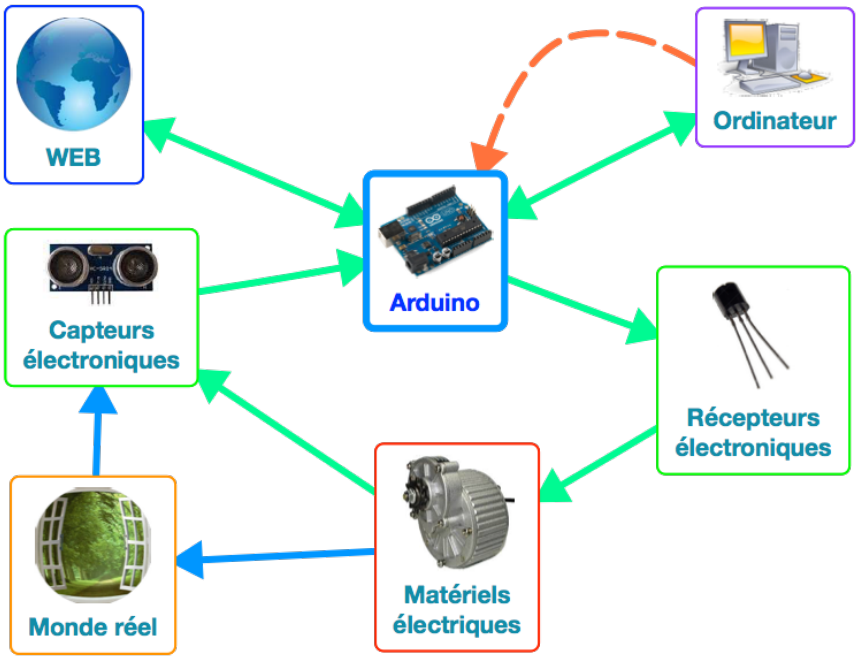
- **Matériel libre :** En fait, les plans de la carte elle-même sont accessibles par tout le monde, gratuitement. La notion de libre est importante pour des questions de droits de propriété.
- **Microcontrôleur :** C'est le cœur de la carte Arduino. C'est une sorte d'ordinateur minuscule (mémoire morte, mémoire vive, processeur et entrées/sorties) et c'est lui que nous allons programmer. Sur la photo précédente, c'est le grand truc

rectangulaire noir avec plein de pattes. Une fois lancé et alimenté en énergie, il est autonome. La force de l'Arduino est de nous proposer le microcontrôleur, les entrées/sorties, la connectique et l'alimentation sur une seule carte. La carte Arduino est construite autour d'un microcontrôleur Atmel AVR (pas toujours le même en fonction de la date de sortie de la carte) avec une capacité de mémoire de 32000 octets pour l'Arduino UNO. Soit 32 Ko, ce qui n'est vraiment pas beaucoup et qui permet pourtant de réaliser un max de projets !



- **Signaux électriques :** L'Arduino fonctionne avec de l'électricité. Un signal électrique est un passage d'électricité dans une partie du circuit. L'Arduino est donc capable de produire ou de capter ces signaux à notre demande grâce à la programmation.

Pour ceux qui se demandent pourquoi ces cartes Arduino ont des noms aux consonances italiennes... et bien la réponse se trouve dans leur origine de fabrication : c'est la société italienne Smart Projects qui crée les modules de base des cartes Arduino ! À noter que quelques cartes sont également fabriquées par SparkFun Electronics, une société américaine. L'Arduino est donc une carte qui se connecte sur l'ordinateur pour être programmée, et qui peut ensuite fonctionner seule si elle est alimentée en énergie. Elle permet de recevoir des informations et d'en transmettre depuis ou vers des matériels électroniques : diodes, potentiomètres, récepteurs, servo-moteurs, moteurs, détecteurs... Nous en ferons une description plus précise au fur et à mesure de ce cours.



Fonctionnement de l'Arduino

Cela va donc sans dire que pour programmer un Arduino, et bien... il vous faut un Arduino ! C'est d'ailleurs l'objet du point suivant, ça tombe bien non ?

Logiciels et matériel pour débuter

On n'a rien sans rien. L'Arduino n'échappe pas à la règle. Il va donc falloir vous équiper de matériel pour bien débuter, et c'est ce que nous allons voir dans cette section !

Petit conseil : à moins que vous n'ayez un distributeur d'électronique près de chez vous, regroupez vos commandes ! Cela vous permettra de limiter les frais de ports qui peuvent parfois coûter plus cher que le matériel...

Matériel

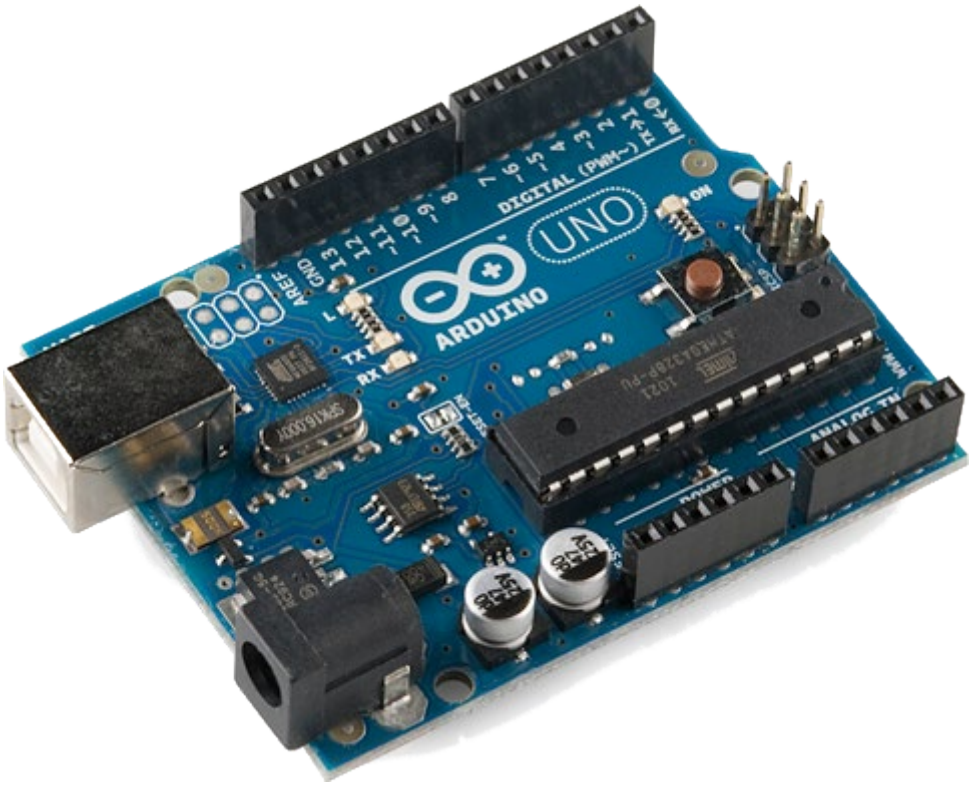
La programmation sur Arduino, du fait de son interfaçage avec des matériels électroniques, nécessite d'acheter du hard...ware ! (Et non Lukas, ce n'est pas ce que vous imaginiez...)
Le **hardware** est la partie matérielle de l'informatique, souvent faite d'électronique, elle s'oppose au **software** qui en est la partie logicielle.

Le coût du hardware nécessaire pour programmer sur Arduino n'est pas excessif, mais il n'est pas anodin non plus. D'autant qu'en fonction des projets que vous souhaitez réaliser, le matériel en jeu peut vous demander d'investir plus que prévu.

Ce cours, au moins pour la première partie, ne va pas demander de casser votre grosse tirelire suisse, mais la petite réservée aux plaisirs exceptionnels (Lukas ça suffit !). Comptez environ 50€ pour bien débuter.

La carte Arduino

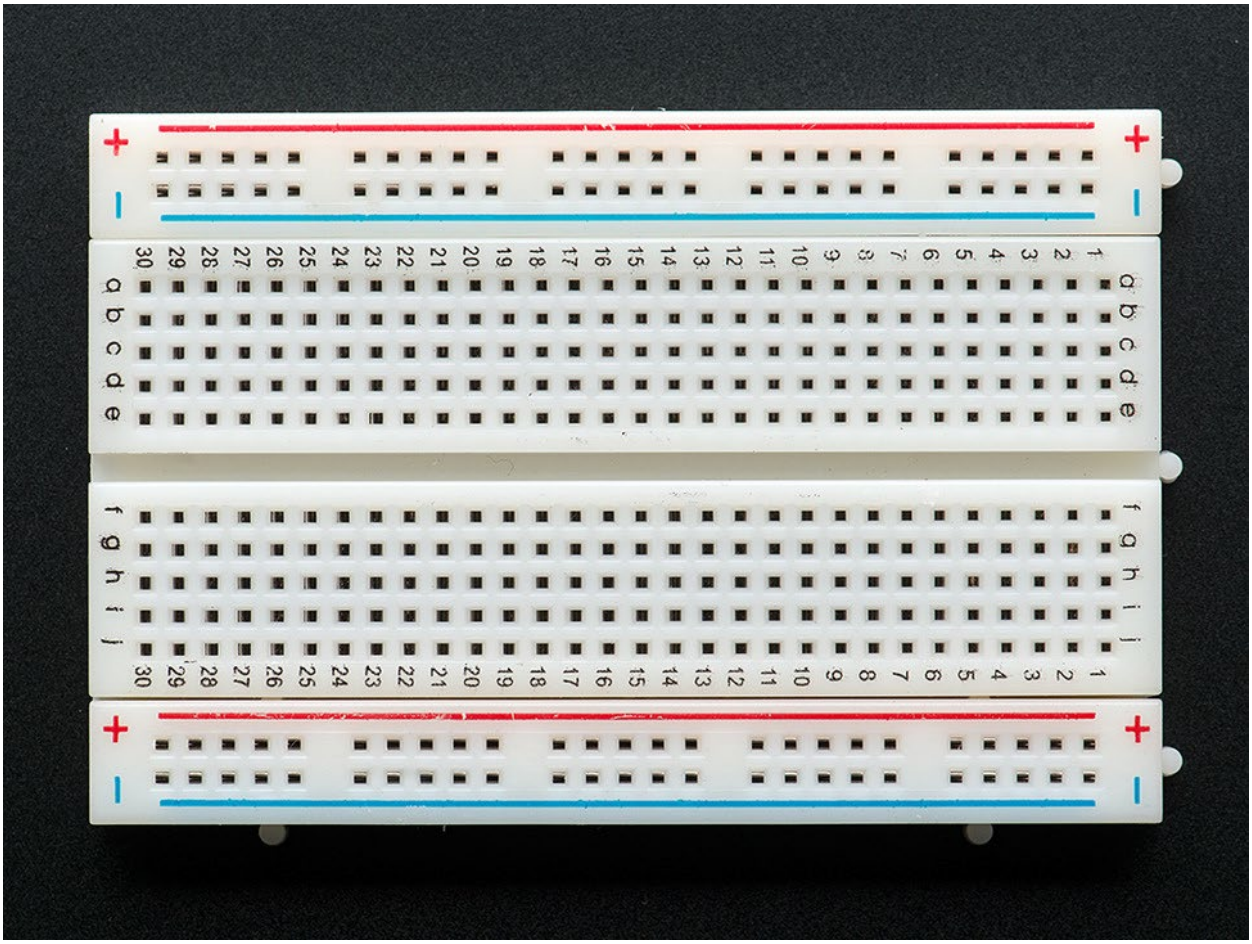
Afin de pouvoir commencer à programmer, il vous faut acquérir **la carte Arduino**. Plusieurs sites de vente de matériel électronique vous la proposent. Ce cours est basé sur l'Arduino UNO R3 que je vous ai montrée dans la section précédente. Vous vous rappelez ? Celle-ci :



Je vous conseille de commencer par cette carte. Si la passion vous gagne, vous aurez rapidement envie (et besoin) d'en acquérir une autre, à ce moment votre choix sera peut-être différent. Le coût est d'environ 25€.

La "breadboard"

Ensuite il faut prévoir une **plaque d'essai** ou **breadboard** qui est indispensable pour tester et interfacer ses projets. C'est une plaque en plastique avec des rangées de trous (par cinq) dans lesquels nous planterons nos composants. J'en explique le fonctionnement plus loin.



Il existe des breadbords de plusieurs tailles. Ne choisissez pas trop petit, en effet vous aurez rapidement besoin de place. Pensez aussi à prendre une breadboard qui comporte deux grandes lignes de trous de chaque côté (voir la photo ci-dessus), ça facilite grandement les connexions.

Les coûts varient entre 5€ et 15€ la plaque. Une seule suffit pour commencer, elle permet de réaliser tous les montages de ce cours. En revanche, là encore, vous aurez très vite envie d'avoir des planches d'essai de diverses tailles en fonction de vos projets. Et si vraiment vous allez plus loin, vous ferez vos propres circuits imprimés, mais nous y reviendrons bien plus tard dans ce cours.

Les fils de connexion

Il vous faudra aussi des fils pour la connexion des composants entre eux et vers l'Arduino : des **jumpers**. Ces fils existent sous plusieurs formats : semi-rigides et dénudés à chaque extrémité, ou bien souples avec une connectique au bout (femelle/femelle, mâle/mâle, mâle/femelle). Vous pouvez bidouiller avec des trombones ou autres trucs métalliques fins, mais réellement, le jumper devient vite indispensable. Là encore on en trouve sur le net et pour 10€ vous serez largement garnis. Voici à quoi ressemble un jumper (ici au format souple) :

Exemple de Jumpers mâle/mâle

Les composants

Enfin, il vous faudra les composants que vous souhaitez contrôler avec votre carte Arduino. Voici la liste que je qualifierais de prioritaire pour suivre les parties 1 et 2 du cours :

- **LED** (pour produire de la lumière) : Au moins 5, vous pouvez panacher les couleurs !
- **Résistances** (pour contrôler le courant) : 5 de 220 Ω, 2 de 10 kΩ, 2 de 1 KΩ.
- **Contacteurs** ou **bouton poussoir** (pour interagir avec le montage) : 2 boutons poussoirs qui peuvent se connecter sur une breadboard.
- **Potentiomètres** (pour créer des variations) : 1 de 10 KΩ, compatible breadboard.

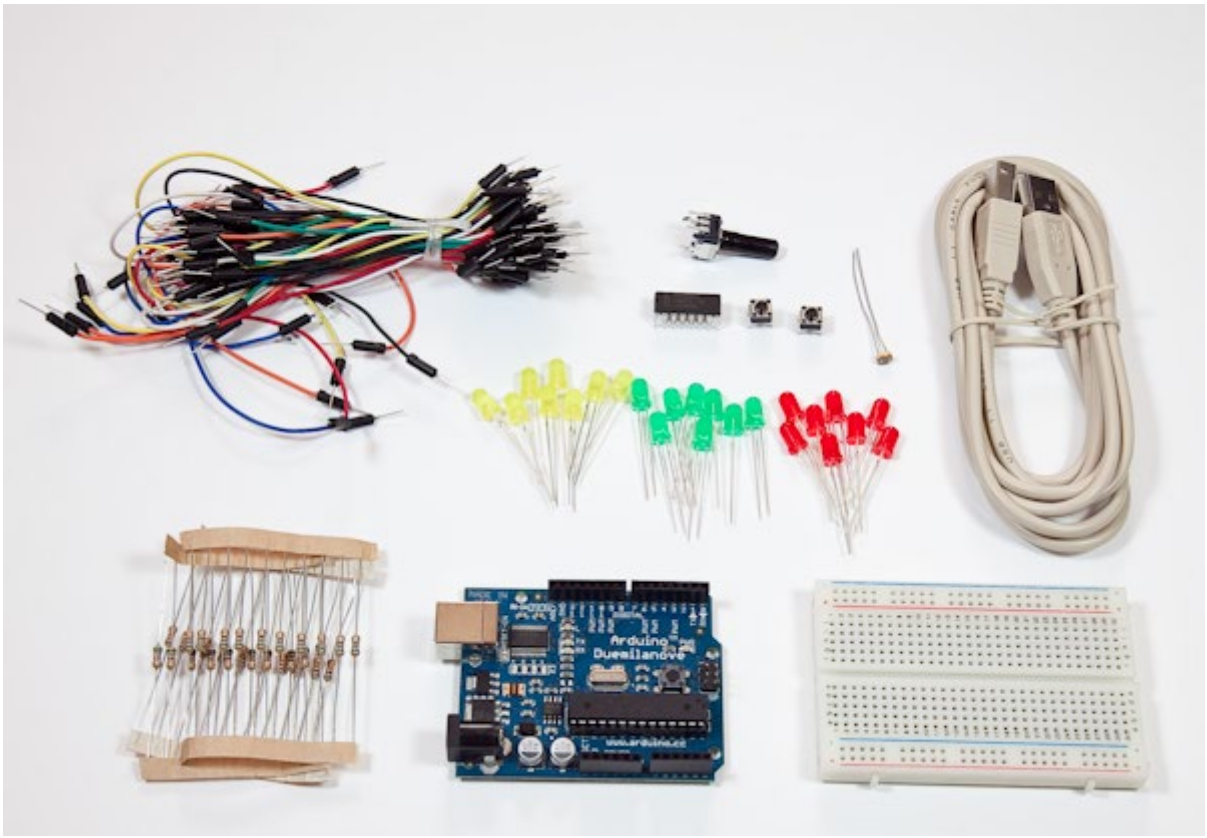
Vous n'en aurez pas pour plus de 10€. Ces composants se trouvent dans n'importe quelle boutique d'électronique, ou en ligne (MCHobby, Adafruit, RSonline, Conrad...).

Voici maintenant une liste qui permet de suivre la suite du cours (dernière partie) :

- **Servo-moteurs** (pour créer des mouvements) : 1 suffira, pilotable sur 5V.

- **Moteur CC** (courant continu) : 2, qui fonctionnent avec une tension entre 3V et 6V.
- Un transistor bipolaire NPN (PN2222) et un MOSFET (IRF540N), Ils servent à piloter les moteurs entre autres.
- Une diode Schottky (1N4001), elle autorise le passage du courant dans un seul sens.
- Une puce L293D, pour piloter les moteur CC.
- etc.

Donc si nous résumons, ça nous fait un total de 55€ à la louche, c'est l'investissement minimum. Je vous conseille de regarder certaines offres qui proposent un kit du débutant ou starter kit aux alentours de 50€ avec le matériel cité au dessus et quelques composants supplémentaires.



Attention, tous les kits de démarrage (starter kit) ne contiennent pas les même composants. Certains sont très complets, et d'autres n'ont même pas la carte Arduino vendue avec. Vérifiez donc bien que le kit contienne au minimum ce que je vous propose, ou équivalent.

Le montant que je vous donne est indicatif, mais un kit très en-dessous de ce prix doit vous mettre la puce à l'oreille...
Enfin, si vous avez la chance d'avoir un magasin d'électronique proche de chez vous, allez y faire un tour, les prix sont légèrement plus élevés mais il n'y a pas de frais d'envoi.

Vous remarquerez le câble USB plat/USB carré (en haut à droite de la photo). C'est le câble type USB PC vers imprimante. Il est nécessaire et n'est pas vendu avec l'Arduino (à part dans un starter Kit), il vous en faudra donc un.
Pour ceux qui sont du genre méticuleux et bien ordonnés, des petites boi-boîtes pour ranger tout ça ne leur paraîtront pas superflues (pour vous Lukas, vous rangerez tout ça dans votre sac à dos entre les sandwichs du mois dernier, vos cours et vos livres illustrés sur le Hardware...)

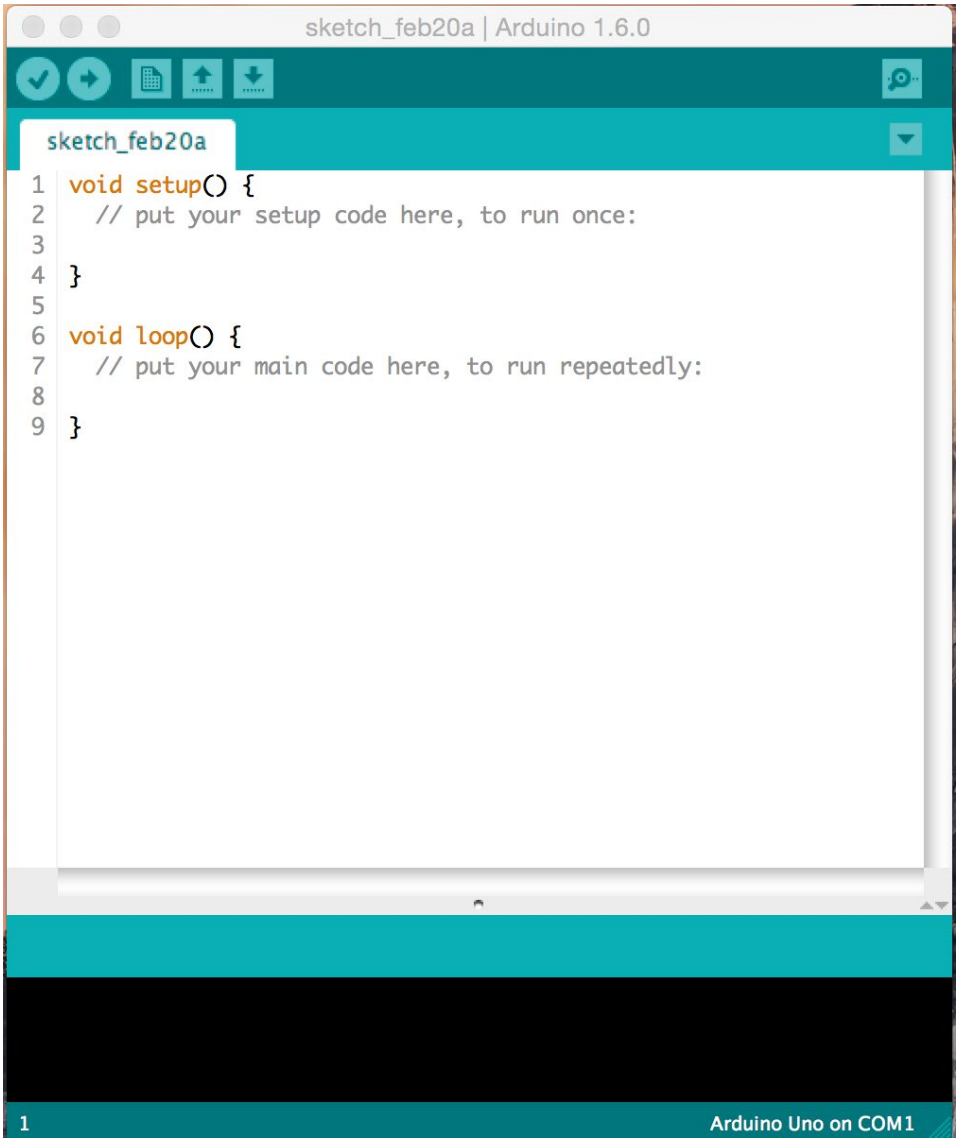
Bon, le hard... c'est fait, le soft maintenant.

Logiciels

Alors la bonne nouvelle c'est que pour le logiciel, il n'y a rien à payer ! (je considère que vous avez déjà un ordinateur bien sûr). Il suffit de télécharger le logiciel sur le site officiel de l'Arduino.

Rendez-vous dans l'onglet "Download", puis en fonction de votre OS (Mac, Windows ou Linux) sélectionnez le bon logiciel et laissez l'installation se faire. Sélectionnez tout de même un emplacement simple et logique pour l'installation. En effet, nous aurons à y revenir par la suite pour y déposer des bibliothèques de codes.

L'installation des paquets sous Linux ne pose aucun problème notable. Je ne m'y étendrai donc pas. Si votre système est sous Linux, je vous propose d'aller sur le **forum du cours** pour échanger à ce sujet si vous rencontrez des difficultés.
Si tout s'est bien déroulé, vous aurez accès à un programme avec un logo vert et un signe infini contenant un plus et un moins. C'est le logo Arduino que l'on retrouve sur la carte. Démarrez le programme et vous obtenez une fenêtre au bords turquoise dans laquelle vous pourrez saisir du texte.



La dernière version du logiciel vous propose une fenêtre déjà remplie du code de base de l'Arduino, nous en parlerons dans le chapitre suivant.

Pour bien des schémas de montages que vous verrez dans ce cours, j'utilise ***Tinkercad***. C'est un site gratuit qui est assez facile de prise en main et qui permet des rendus clairs. Bien que je n'en expliquerai pas l'utilisation dans ce cours, je vous conseille de l'essayer car il vous permettra de mieux organiser vos projets plus complexes.
Vous êtes maintenant prêt(e)s pour la suite.

Encore une dernière petite section avant de cheminer dans le monde sibyllin de la programmation de la carte Arduino (Lukas, vous irez chercher sibyllin dans le dictionnaire ça vous occupera...)

Organisation du cours

Vous vous apercevrez rapidement que la communauté libre qui gravite autour de l'Arduino est riche et dynamique. On trouve de toutes parts des forums, des sites, des blogs qui regorgent d'exemples, d'idées, d'aides et de conseils. Ce cours n'est pas là pour balayer l'existant, loin de là.

Le but est de vous faire découvrir progressivement la programmation sur Arduino et les connaissances électroniques qui vont avec. Je pars du principe que chacun est libre d'y puiser ce qui lui convient, mais si vous êtes complètement novices, vous y trouverez un moyen efficace et régulier pour apprendre, comprendre et progresser à un rythme qui me semble suffisant du fait de l'ensemble des connaissances à acquérir.

Il est évident qu'un développeur de niveau moyen va trouver la programmation facile et s'intéressera plus à la partie électronique, et à l'inverse pour un fan d'électronique. Mais l'un comme l'autre pourra progresser et s'ouvrir à d'autres domaines, comme la mécanique par exemple lorsqu'il devra faire avancer son robot.

Chaque chapitre débute par ses objectifs, cela permet ainsi à chacun de retrouver rapidement les notions abordées. J'ai cherché ensuite, dans la mesure du possible, à toujours proposer une activité pratique pour faire suivre la théorie.

Vous trouverez également plusieurs activités pratiques pour vous approprier la théorie. Et oui, la programmation sur Arduino ne déroge pas à la règle du : "essaie par toi-même, transforme et modifie, pour mieux t'emparer du savoir"(non Lukas, ce n'est pas une citation tirée de Star Wars, c'est de moi). En fait il n'y a aucun risque, sinon celui de mieux comprendre, à modifier un programme proposé. Vous verrez d'ailleurs que le logiciel de l'Arduino en propose des tous prêts qui sont riches d'enseignements et qui vous serviront sûrement de base pour la suite de vos projets.

L'important est de comprendre ce que l'on fait et pourquoi on le fait.

En revanche, sur la partie électronique, il ne faut pas s'amuser à brancher tout n'importe où. Nous le verrons par la suite, l'Arduino est parcouru par du courant électrique, et un branchement mal fait peut le faire griller. Donc pour ce cas seul, des précautions s'imposeront.
Bon, je sens que vos doigts fourmillent d'impatience et que votre intellect en soif d'apprentissage se retient par la seule force de votre savoir-vivre de me crier : « mais il va passer à l'action oui ou non ? ! ».

Je comprends, alors attaquons !

2. Créer votre premier programme

Créez votre premier programme sur Arduino

Dans ce chapitre, je vais aborder la base de la base du début du commencement. C'est-à-dire que ce qui va être dit là est incontournable et nécessaire. J'ai l'air d'insister un peu ? Tout à fait. Car une fois cette première étape passée, vous aurez mis un pied dans la programmation Arduino. Et comme lorsqu'on se lève le matin, il est préférable que ce soit sur le bon pied !

Nous allons commencer par observer la structure de base d'un programme et du même coup faire un tour du fonctionnement de l'interface de programmation.

Nous découvrirons ensuite la LED 13 et nous apprendrons à la faire clignoter.

Ce chapitre sera aussi l'occasion de faire un rapide survol du principe de compilation et du fonctionnement général du microcontrôleur.

Si tout cela vous paraît facile, tant mieux, mais vous allez voir que notre premier programme va nous réserver quelques surprises...

La structure de base d'un programme Arduino

Bon, il faut savoir que le langage utilisé par le logiciel Arduino pour programmer le microcontrôleur est basé sur les langages C/C++.

Vous allez de ce pas créer les premières lignes de votre premier programme Arduino. À partir de maintenant, il va falloir admettre que si votre programme plante, ce n'est pas la faute de l'ordinateur ou de l'Arduino, mais la vôtre. En effet, un ordinateur, même de la taille de l'Arduino, va vous obéir au doigt et à l'oeil, mais jamais ne réfléchira par lui-même. Donc si la série d'instructions que vous lui donnez l'amène à sa destruction, et bien il se détruira. Ou au mieux, il se bloquera. Nous allons donc devoir apprendre à éviter ces situations énervantes que provoquent les bugs.

Mais voyons plutôt notre premier programme. Je vais vous demander d'ouvrir le logiciel Arduino, pour le moment nous n'allons pas connecter la carte à votre ordinateur. Une fois la fenêtre d'accueil ouverte, vous allez taper dans la zone blanche ce qui suit (soyez rigoureux et faites attention aux ponctuations et majuscules) :

```
void setup()

{



}
```

```
void loop()


{




}
```

Voici ce que ça donne sur votre console de programmation :

```
1  void setup()
2  {
3
4  }
5
6  void loop()
7  {
8
9  }
```

Pour ceux qui ont la dernière version d'Arduino (version 1.6.5 au mois d'octobre 2015), ces lignes apparaîtront par défaut dans la fenêtre, je vous demanderai de tout effacer et d'écrire à partir de la page blanche, c'est pour mieux comprendre la suite.

Alors vous avez sûrement remarqué que les mots se colorent lorsqu'on les a saisis. C'est un petit plus très pratique qui s'appelle la **coloration syntaxique**. Elle très employée en informatique. Elle nous permet de faire la différence entre les mots clés du langage utilisé, et les autres.

Pour les artistes sensibles de la pupille du genre :

Je suis désolé de leur annoncer que nous n'avons pas le choix de la couleur, mais ça reste très pratique !

Un programme informatique est une succession d'instructions que l'on donne à la machine. Ces instructions sont écrites dans un langage qui utilise des mots spécifiques. Savoir programmer, c'est déjà connaître les mots dont on a besoin.
Je ne vais pas ici donner la liste de tous les mots utilisés par le logiciel Arduino, nous les apprendrons lorsque nous en aurons besoin. Mais dans le programme au-dessus, vous venez d'en rencontrer trois : **void**, **setup()**, **loop()**. Retenez-les !

Regardons à nouveau le programme, nous observons des parenthèses et des accolades. Pour les utilisateurs de Windows, ça reste assez évident à obtenir. Pour ceux qui ont un Mac, l'accolade s'obtient en utilisant la combinaison : `alt`+`(` ou `alt` +`)`.
Vous verrez bientôt dans ce chapitre l'utilité des parenthèses et des accolades. L'important pour l'instant, c'est de retenir que dans un programme écrit, toute parenthèse ou accolade

ouverte DOIT être refermée. Cet oubli est souvent la cause de plantage du programme. Nous verrons plus tard qu'elles peuvent s'imbriquer les unes dans les autres. Je vous conseille donc d'ouvrir une accolade, de la refermer juste après et de revenir entre les deux pour ajouter votre code.

Le menu du logiciel

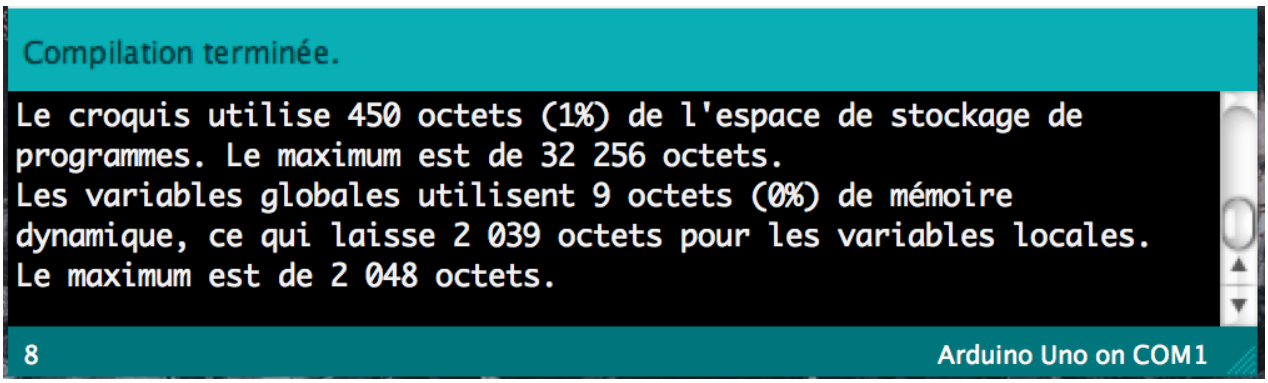
Intéressons-nous maintenant au menu proposé par le logiciel :

Ces cinq boutons sont quasiment les seuls que nous aurons à utiliser pour lancer le programme (nous verrons d'autres fonctionnalités par la suite). Nous allons donc les étudier un par un dans ce chapitre. Vous remarquerez qu'en les survolant, un texte d'aide s'affiche sur la droite des icônes.

Le premier bouton à gauche : le V de vérifier.

Ce bouton permet de vérifier votre programme. C'est-à-dire que le logiciel Arduino va chercher si ce que vous avez écrit est conforme à ce qui est attendu.

Allez-y, cliquez dessus. Si vous avez bien écrit les lignes que je vous ai demandées dans la section précédente, vous obtenez en bas, dans la fenêtre noire, un texte du genre :



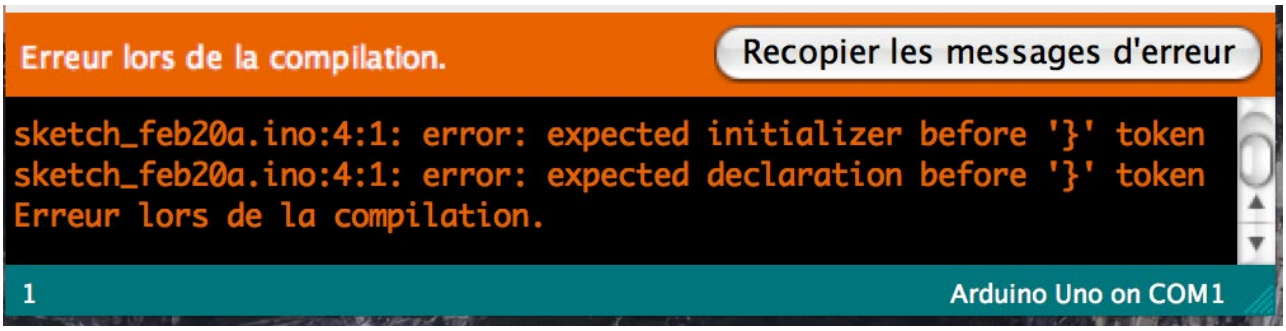
Ça signifie que le programme est correct.

J'utilise une version française du logiciel Arduino. C'est plus pratique. Comme dans les dernières versions, elle est disponible automatiquement, je ne traduirai pas la version anglaise.

Qu'est-ce que la compilation ?

De le savoir ne vous changera pas la vie, mais je comprends les curieux. En fait le programme que vous écrivez n'est pas celui que recevra l'Arduino. C'est un programme adapté pour que les humains le lisent et le comprennent sans trop de difficulté. La machine, elle, ne comprend que le **langage machine**, une succession de 1 et de 0, qu'elle va interpréter et exécuter. La compilation est la transformation du langage de programmation utilisé par l'homme, en langage machine utilisé par la machine. C'est d'ailleurs le rôle de l'interface pour l'Arduino.

Vous pouvez maintenant essayer de modifier votre programme en enlevant la première accolade, vous cliquez la vérification et vous obtenez en bas un message du genre :



On sent tout de suite à la couleur qu'il se passe quelque chose de pas net, et on a raison !

Le logiciel de l'Arduino propose aussi un petit débuggeur. Il ne va pas vous réparer tout seul l'erreur, mais il vous indique où il pense qu'elle se trouve. C'est ce qui se passe dans notre cas. Pour comprendre un peu ce qui ce passe, il faut lire le message d'erreur :

- "sketch_feb20a.ino" c'est le nom de votre programme.
- "4:1" signifie que ligne 4 caractère 1, le programme bloque pour la compilation.
- "expected" un truc, indique le type d'erreur. Mais ce n'est pas toujours la bonne raison.

Les erreurs fréquentes sont que vous avez oublié de taper quelque chose ou que vous n'avez pas respecté le format attendu. Dans les premiers temps, vous oublierez surtout des points-virgules et des fermetures d'accolades ou de parenthèses. Plus tard, les problèmes se corseront car le logiciel ne verra pas forcément d'erreur et pourtant votre programme ne fera pas ce que vous souhaitez. Mais nous n'en sommes pas là. (Je vois à votre tête décomposée Lukas que la perspective de réfléchir sans aide vous paraît inaccessible, rassurez-vous, tout le monde finit par y arriver !)

Pour réparer l'erreur dans notre cas, remettez l'accolade, puis cliquez à nouveau sur le bouton Vérifier. Tout devrait rentrer dans l'ordre.

Je vous conseille de vous amuser dès le début à créer des erreurs volontaires. Elles vous permettront de mieux comprendre le fonctionnement du débuggeur et vous dérouteront moins par la suite.
Passons à l'icône suivante :

La flèche de téléverser

MESDAMES ET MESSIEURS, JE VOUS ANNONCE OFFICIELLEMENT QUE VOUS POUVEZ CONNECTER LA DERNIÈRE MERVEILLE TECHNOLOGIQUE, J'AI NOMMÉ L'ARDUINO, À VOTRE ORDINATEUR PERSONNEL GRÂCE AU CÂBLE USB !

Et là, c'est pas gagné pour tout le monde. En effet en fonction de votre système, l'Arduino va ou non être reconnu. Sur Windows, vous devrez peut-être attendre qu'il installe un pilote de périphérique. Il n'y a pas trop le choix, il faut le faire.

J'ai rencontré ce cas sur Windows uniquement : si vous changez de port USB, il va vous demander d'installer à nouveau le pilote. Vous avez deux choix, soit vous décidez de ne connecter l'Arduino qu'au même port USB, soit vous les faites tous un par un jusqu'à ce que Windows ait tout installé.
Une fois connectés, vous pouvez cliquer sur l'icône avec la flèche pour téléverser le programme. Le logiciel va donc transférer votre programme compilé (transformé en langage machine) dans la mémoire du microcontrôleur de l'Arduino. Une fois cette opération effectuée, l'Arduino gardera ce programme en mémoire et l'exécutera tant qu'il sera alimenté en électricité. Il sera donc autonome et ne dépendra plus de l'ordinateur !

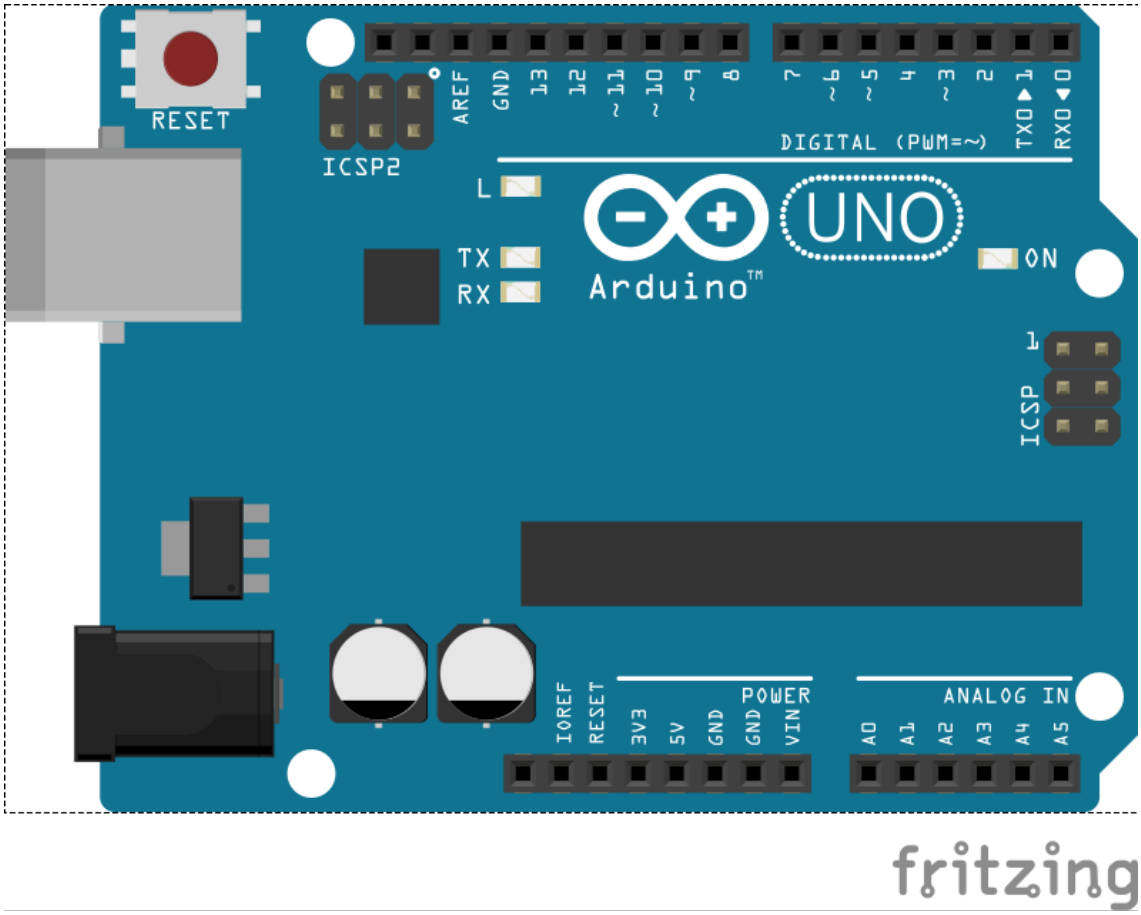
Certains vont rencontrer un problème. Il arrive fréquemment, il faut savoir s'en dépêtrer : l'ordinateur n'arrive pas à transférer les données car la connexion ne se fait pas avec l'Arduino. Pas de panique !

Vous avez dans les menus du logiciel (en haut) un menu "Outils", dans ce menu, vous trouverez un sous-menu "Port Série" qui vous propose plusieurs ports à utiliser. Changez de port sélectionné et réessayez jusqu'à trouver le bon. Vous aurez peut-être à refaire cette opération d'autres fois. Elle est assez récurrente sur Windows.
Mais au fait, il fait quoi ce programme qu'on vient d'écrire ?
Et bien il ne fait rien ! Strictement rien ! Pas facile, essayez vous-mêmes, on fait toujours un truc. Là ça ne fait rien ! Sauf que c'est l'ossature même de TOUS les programmes que nous ferons par la suite. **C'est le code de base.** Quel que soit le programme, on commence par taper ça, et ensuite on remplit ce qu'il faut où il faut.

Maintenant que vous connaissez la forme de base d'un programme, attaquons-nous à du lourd ! Construisons un programme qui fait quelque chose.

La LED 13

Puisque nous avons la carte Arduino sous les yeux, observons-la un peu.



Cette image reprend les parties importantes de la carte. Nous allons nous intéresser au haut de cette image (orientez votre Arduino pour que ça corresponde). Vous voyez des trous numérotés de 0 à 13, puis GND et AREF.

Un peu en dessous, vous voyez 3 petits rectangles notés L, TX et RX, et un autre à droite noté ON.

Ces rectangles sont des LED, c'est-à-dire des "**L**ight-**E**mitting **D**iodes". Nous verrons plus tard ce que c'est, pour le moment il suffit de comprendre que ça peut s'allumer. Ici nos 4 LEDs peuvent briller dans plusieurs cas :

- La LED **ON** est verte quand l'Arduino est sous tension.
- Les LEDs **TX** et **RX** clignotent quand l'Arduino reçoit ou envoie des informations.
- La LED **L** clignote si on appuie sur le bouton reset, sinon elle ne fait rien tant qu'on ne lui a pas dit.

C'est la LED **L** que nous allons allumer grâce à notre petit programme. Une diode s'allume quand elle est parcourue par le bon courant, dans le bon sens.

La force de l'Arduino (entre autre) est d'envoyer du courant, ou non, par les connexions numérotées de 0 à 13. Et la diode L s'allume quand on dit à l'Arduino d'envoyer du courant dans la connexion 13. C'est pour ça que je l'appelle la LED 13.

Vous verrez qu'elle est bien pratique et qu'elle peut servir dans certains cas pour nous aider à débugger. Savoir l'allumer et l'éteindre à volonté est une opération de base sur l'Arduino.

Les étapes du programme sont assez simples :

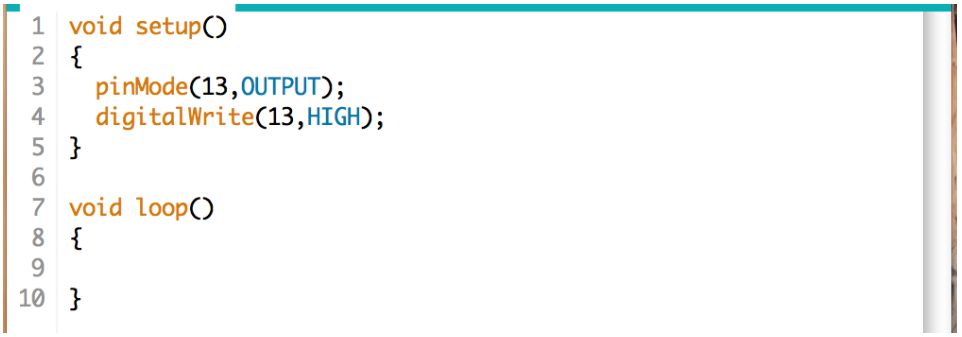
1. On dit à l'Arduino que nous voulons que la connexion 13 puisse envoyer du courant (et pas en recevoir).
2. On dit à l'Arduino d'envoyer du courant dans la connexion 13.

Programmer l'Arduino c'est transformer ces instructions en langage codé. Et ça vous ne pouvez pas le deviner seuls.

Voici donc le programme à taper et à envoyer à l'Arduino (attention à nouveau à la ponctuation et aux majuscules !) :

```
void setup()

{

    pinMode(13,OUTPUT);  //Signale à l'Arduino que la connexion 13 doit pouvoir envoyer du
courant

    digitalWrite(13,HIGH);  //Demande à l'Arduino d'envoyer du courant dans la connexion
13

}




void loop()

{




}
```

Et voici le rendu coloré du logiciel Arduino :



Programme pour allumer la LED 13

Le point-virgule a une importance primordiale ! Il indique à l'Arduino la fin d'une instruction. Si vous l'oubliez (et vous l'oublierez parfois) le programme ne pourra pas se compiler.
J'attire votre attention sur des nouveaux mots-clés : **pinMode()**, **digitalWrite()**, **OUTPUT**, **HIGH**.

Ils ne sont pas colorés pareil, nous verrons plus tard pourquoi. Vous observez que la structure de tout à l'heure n'a pas changé, on a juste ajouté entre les accolades deux instructions.

Le nombre 13 n'est pas anodin, c'est bien la connexion 13 qui est en jeu.

Ben c'est bien beau tout ça, mais comment on l'éteint cette diode maintenant ?
On ne peut pas l'éteindre. Même si vous appuyez sur le reset de l'Arduino, la diode se rallume. Il fait ce qu'on lui a demandé. Pour l'éteindre, il faut donc lui envoyer un nouveau programme qui va lui dire d'éteindre la diode.

Les mots clés en bleu dans le logiciel, sont spéciaux. Ils s'écrivent en majuscules. Ce sont des valeurs que l'Arduino connaît et utilise. Dans notre cas, le mot clé **HIGH** (Lukas, on prononce plutôt "hhaï" que "igue") signifie "valeur haute". Il existe son contraire : **LOW** qui signifie "valeur basse". On pourrait aussi dire que HIGH vaut 1 et LOW vaut 0, ou encore que HIGH veut dire que la connexion est à +5V et LOW, qu'elle est à est à 0V.

Je vous laisse modifier le programme (et le téléverser) afin que la diode s'éteigne. Il est important d'essayer seul de chercher la solution, de débugger et de finalement trouver. Regarder la correction sans avoir réfléchi par soi-même avant est beaucoup moins formateur.

Ça y est, vous avez trouvé la solution ? Alors vérifiez la correction !

Vous avez trouvé cela un peu long jusqu'à maintenant ? Faites une petite pause si besoin : un petit café, un petit tour dehors... ce qui vous plaît pour vous changer les idées 5 minutes ! (Non Lukas, cette proposition est hors-sujet)

Lorsqu'on débute vraiment (je ne parle pas de ceux qui ont déjà touché du code) ces concepts peuvent paraître abstraits et déroutants. L'expérience vous aidera par la suite à mieux créer des liens entre tout ce que vous aurez ingurgité.

Bon si vous êtes prêts, nous allons attaquer notre premier programme plus compliqué. On a allumé la diode, on l'a éteinte, et si on la faisait clignoter maintenant ?

Votre premier programme : «Blink Blink !»

À chaque fois que vous décidez de réaliser un programme, il est important de savoir à l'avance, le plus précisément possible ce que vous voulez obtenir. Il est toujours tentant de se lancer directement dedans et de modifier au fur et à mesure, mais plus votre programme sera complexe, plus le modifier pour intégrer des nouvelles possibilités sera compliqué. Dans notre cas, ça devrait aller. Ce programme va nous permettre de mieux comprendre la structure de base du programme Arduino et la raison des deux séries d'accolades.

Ouvrez le logiciel pour l'Arduino et cliquez sur l'icône de nouveau programme et nommez-le "BlinkBlink".

Nous voulons faire clignoter la LED 13 donc les étapes seraient :

1. Nous indiquons à l'Arduino que la connexion 13 doit pouvoir envoyer du courant
2. Nous indiquons d'envoyer du courant dans la connexion 13 (la diode s'allume)

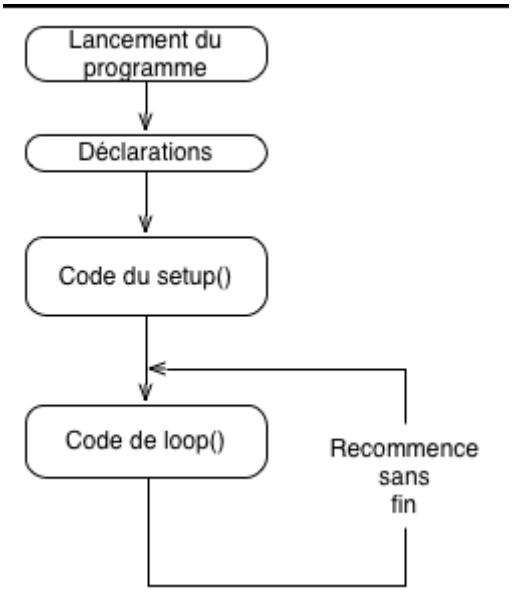
3. Nous indiquons d'arrêter d'envoyer du courant dans la connexion 13 (la diode s'éteint)
4. Nous voulons ensuite recommencer au point numéro 2 à l'infini.

Les points 1, 2 et 3 vous pouvez maintenant les programmer sans problèmes. Là où ça se complique, c'est le point 4.

Et là l'Arduino, et sa structure de base, nous facilitent la tâche. En effet les accolades sont des sortes de capsules. Tout ce qui est entre les accolades est considéré comme un bloc d'instructions. Nous avons dans le programme deux sortes d'accolades : celles qui suivent l'instruction`setup()` et celles qui suivent l'instruction`loop()`.
Jusqu'à maintenant, nos instructions ont été mises dans les accolades de "setup". Nous dirons qu'elles sont dans le setup. Ces instructions ne sont lues et activées qu'**une seule fois.**

Les instructions mises dans les accolades de "loop" seront lues et activées **à l'infini** !

Ce qui donne en représentation graphique :



Prenons l'exemple du pseudo-programme suivant :

```
void setup()

{

    instruction 1;

    instruction 2;

}



void loop()
```

```
{

    instruction 3;

    instruction 4;

}
```

Voici le cheminement du programme :

Ligne	Ce que fait le programme
1	Il se prépare à lire le bloc setup()
2	Il repère le début du bloc setup()
3	Il exécute l'instruction 1
4	Il exécute l'instruction 2
5	Il repère la fin du setup()
6	Pas d'influence sur le programme
7	Il se prépare à lire le bloc loop() à l'infini
8	Il repère le début du bloc de la loop()
9	Il exécute l'instruction 3
10	Il exécute l'instruction 4
11	Il rencontre la fin du bloc loop() donc il recommence au début du bloc (en 8)
9	Il exécute l'instruction 3
10	Il exécute l'instruction 4
11	Il revient en 8, etc.

Et cela jusqu'à l'infini ou presque, ou tant qu'il aura du courant.

Ce tableau vous paraît sûrement bien long. C’est fait exprès ! Il permet de bien différencier une exécution unique (setup) d'une exécution en boucle (loop), de plus il rappelle que les accolades montrent des limites de blocs d'instructions de programme. C'est à retenir absolument pour la suite.

Alors du coup ça change un peu ce qu'on doit écrire si l'on veut que ça clignote à l'infini. Je vous laisse chercher un peu...

Voici la solution que vous pourriez proposer dans l'état actuel de vos connaissances :

```
void setup()

{

    pinMode(13,OUTPUT);

}



void loop()

{

    digitalWrite(13,HIGH);

    digitalWrite(13,LOW);

}
```

Si vous avez trouvé cette solution seul(e)s, c'est que vous avez compris.

Téléversez et observez !

Et là... c'est le drame ! En effet, tout est ok, Arduino exécute la première instruction une fois dans le setup, puis la loop exécute le reste à l'infini : j'allume, j'éteins, j'allume, j'éteins. Et pourtant la diode reste allumée !

Voilà exactement la situation la plus délicate pour un programmeur : il n'y a pas de bug, tout paraît nickel et pourtant le résultat n'est pas là. Alors que faire ?

Dans votre cas vous êtes juste victimes du léger sadisme de celui qui vous guide dans ce cours, c'est à dire MOUA. En effet je ne vous ai pas tout dit, mais il fallait en passer par là. Nous allons devoir aborder un peu de physique (non Lukas, je parle de sciences physiques) et plus particulièrement les notions de fréquences et de périodes. Ne vous alarmez pas, ça coule tout seul.

Fréquence et période

Quand on se penche un peu sur la documentation de l'Arduino, on peut lire :

"Chaque module possède au moins un régulateur linéaire 5 V et un oscillateur à quartz 16 MHz (ou un résonateur céramique dans certains modèles)."

Ça veut dire que d'une part, la tension de sortie au niveau des connexions est de cinq volts (nous y reviendrons plus tard) et que notre processeur est cadencé (j'adore ce mot) à 16 MHz (seize méga hertz)

Alors j'entends déjà vos cloches à questions sonner et dire :

C'est quoi un méga hertz ?

Et bien dites-vous que c'est simple (ça met dans un bonne disposition mentale) car le **hertz** c'est l'unité de mesure de la **fréquence** (comme le mètre est l'unité de mesure de la distance).

Et la **fréquence** c'est juste le nombre de fois par seconde qu'une action est faite. Par exemple si je tape 10 touches par seconde sur mon clavier, ma fréquence de frappe est de 10 hertz, on note 10 Hz.

Voici quelques exemples de fréquences :

Exemple	Fréquence	Action
Respiration	0,5 Hz	1 respiration toutes les 2 secondes
Seconde	1 Hz	1 seconde toutes les secondes (je précise,on ne sait jamais)
Pouls humain	1,2 Hz	1,2 battement par seconde soit 72 par minute environ
Courant alternatif	50 Hz	50 oscillations par seconde
La du diapason	440 Hz	440 oscillations par seconde
Ondes FM	entre 23 et 95 KHz	entre 23 000 et 95 000 vibrations par seconde
Arduino	16 MHz	soit 16 000 000 d'actions machine par seconde
Processeur actuel	3 Ghz	soit 3 000 000 000 d'actions machine par seconde

Ça donne une bonne idée de ce qu'est un Hertz (Lukas, si vous continuez de m'encombrer l'appareil auditif avec votre marque de location de voiture, je vous colle une fessée cadencée à 3 Hz dont vous me direz des nouvelles !)

Si notre Arduino est bien moins rapide que des processeurs actuels, il est bien plus rapide que nous !

16 000 000 d'actions par seconde ne veut pas dire 16 000 000 d'instructions par seconde. Une action machine est un passage ou non d'un courant électrique (1 et 0). Une instruction nécessite une multitude d'actions machine pour se réaliser. Les puristes diront que je simplifie, mais c'est pour donner une idée générale.
Dans le cas de notre programme il clignote à un peu moins de 77 KHz, soit 77 000 fois j'allume/j'éteins par seconde. Notre œil ne perçoit des images différentes qu'en dessous de 25Hz.

Pour les curieux qui se demandent comment je sais que notre LED clignote à environ 77 Khz, le calcul de la vitesse d'exécution d'une instruction peut être réalisé avec l'horloge interne de l'Arduino. Le principe est simple : on stocke dans une variable le temps actuel, on exécute l'instruction (ou le bloc d'instructions) dont on veut mesurer la vitesse d'exécution, et on fait la différence entre le temps actuel et le temps stocké et on l'affiche sur la console.

```
void setup() {

  Serial.begin(9600);

}

void loop() {

  unsigned long tempsFin; //variable pour noter le temps final

  unsigned long  tempsDepart=micros(); //variable du temps de départ initialisée au temps
actuel

  //bloc à tester

  tempsFin=micros(); //on récupère à nouveau le temps actuel

  Serial.println(tempsFin-tempsDepart); // on affiche la différence en microsecondes

}
```

Je vous donne ce code à titre indicatif, c'est tout à fait normal si ça vous paraît encore obscur, mais n'hésitez pas à tester vous-mêmes si ça vous intéresse.
Solution : pour voir le clignotement, il faut ralentir l'Arduino.

Nous n'avons pas le choix, nous devons apprendre un nouveau mot clé : **delay()**.

delay() indique à l'Arduino d'attendre. On a juste à mettre entre parenthèses le nombre de millisecondes que l'on souhaite. Ça nécessite un léger calcul mental : 1 s = 1000 ms.

Par exemple, l'instruction suivante :

```
delay (500);
```

met le programme en pause pendant 500 ms=0,5 s donc une demi-seconde.

On utilise donc l'instruction`delay()` en écrivant la ligne de code où l'on veut que le programme fasse une pause. Je vous laisse essayer de modifier le programme pour que notre LED 13 s'allume 1 seconde et s'éteigne 1 seconde.
Et voici le corrigé :

```
void setup()

{
```

```
  pinMode(13,OUTPUT);

}



void loop()

{

  digitalWrite(13,HIGH);

  delay(1000);

  digitalWrite(13,LOW);

  delay(1000);

}
```

Amusez-vous maintenant à modifier les valeurs des`delay()` pour obtenir des clignotements différents.

Les commentaires dans un programme

Tous les programmeurs, ou codeurs, ont des conventions à respecter s'il souhaitent que leur travail soit considéré comme agréable et bien fait. Nous verrons au fil de ce cours quelques unes de ces conventions, et la première dès maintenant :

Un programme se doit d'être commenté.

C'est-à-dire que lorsque l'on code, il est important, autant pour soi-même que pour ceux qui liront ou utiliseront votre travail, de donner des indications sur le fonctionnement du programme, l'utilité des variables et des fonctions (nous en parlerons plus loin) et bien d'autres choses.

Pour insérer un commentaire simple, il suffit de saisir le code // (double slash) et du coup, tout ce qui suivra dans la ligne sera lisible par un humain et transparent pour le programme.

Pour un bloc de commentaires (c'est-à-dire des commentaires sur plusieurs lignes à la suite) on utilise deux balises : /* et */

Voici donc notre programme avec des commentaires :

```
/*
*****************************
programme Blink Blink
Auteur Nanomaitre
Pour openClassRoom
02/2015
Libre de droits
*****************************
*/

void setup() // boucle d'initialisation
{
  pinMode(13,OUTPUT); //connexion 13 en mode envoi de courant
}

void loop() // boucle inifie
{
  digitalWrite(13,HIGH); //envoi de courant dans la connexion 13
  delay(1000); //attente d'une seconde
  digitalWrite(13,LOW); //arret du courant dans la connexion 13
  delay(1000); //attente d'une seconde
}
```

Les commentaires proposés ici présentent un niveau de détail excessif. C'est pour vous montrer comment on utilise les balises //, /* et */. Essayez de commenter votre programme et prenez l'habitude de le faire en particulier pour les programmes plus complexes, ce n'est pas du temps perdu.
Un dernier point concernant cette partie avant de clore ce chapitre...

Sauvegarde et transfert de votre programme

Lorsque vous téléversez votre programme, il reste stocké dans le microcontrôleur. Vous pouvez débrancher ce dernier, le brancher à une pile 9V via la prise prévue à cet effet, ou à un autre ordinateur (sans lui envoyer de programme) et vous verrez que le programme que nous venons de concevoir tourne encore. Il est donc sauvegardé.

Mais il n'est plus accessible !

C'est-à-dire que vous ne pouvez pas faire l'opération inverse qui consisterait à sortir le programme de l'Arduino pour le mettre sur le logiciel de votre ordinateur.

C'est la raison pour laquelle il vous faut sauvegarder votre travail. C'est le bouton le plus à droite (une flèche vers le bas) proposé par le logiciel :

D'ailleurs, la dernière version propose la sauvegarde avant même de compiler.

Vos fichiers sont stockés dans un dossier accessible depuis le logiciel par le menu en haut : on choisit "Fichier" puis "**Carnet de croquis**" et on voit apparaître ses sauvegardes. On retrouve le même dossier dans le dossier où vous avez installé l'Arduino.

Pour l'ouvrir, vous pouvez utiliser l'autre bouton (flèche vers le haut), qui vous ouvre le menu dans lequel vous verrez dans notre cas "BlinkBlink".

Mais que sont donc tous ces autres croquis proposés : 01.Basics, 02.Digital,... ?
Si vous vous posez cette question, c'est que vous avez ouvert le dossier "**Exemples**" qui se trouve en dessous de "Carnet de croquis". C'est bon signe, c'est que vous commencez à prendre en main le logiciel (qu'on appelle aussi IDE pour Integrated Development Environment, c'est-à-dire environnement de développement intégré). Ces autres croquis sont la force de l'IDE, en effet, ce sont des exemples de programmes proposés par Arduino (ou parfois par les concepteurs de bibliothèques) pour nous aider à utiliser correctement les mots clés du langage. Ce sont en fait des exemples. Je vous invite à les parcourir. C'est souvent commenté en anglais, mais il va falloir s'y habituer car la langue de Shakespeare est la base de la communication entre développeurs.

Vous verrez que rapidement, avec ces exemples et la documentation du site officiel (dont nous parlerons plus tard) vous volerez vite de vos propres ailes !

En résumé

Vous savez maintenant connecter votre Arduino, écrire la structure de base d'un programme, sauvegarder et transférer le programme dans le microcontrôleur, et débugger certaines erreurs de code.

Vous avez appris les mots-clés :

- `setup()` qui s'exécute qu'une fois ;
- `loop()` qui s'exécute à l'infini ;
- `void` (qui se met toujours devant loop et setup et dont vous comprendrez le sens plus loin dans ce cours) ;
- `pinMode()` qui permet de définir une sortie en mode envoi d'électricité ou non ;
- `digitalWrite()` qui envoie de l'électricité par une sortie ;
- `delay()` qui met le programme en pause pendant un nombre défini de millisecondes.

Vous savez aussi que les **accolades** délimitent des blocs d'instructions et que le **point-virgule** termine une instruction.

Vous savez enfin que la connexion 13 est reliée à une **LED** (diode électroluminescente), que vous pouvez maintenant allumer ou éteindre.

Je pense que vous êtes prêt(e)s pour la suite et je vous en félicite !

(Quelqu'un peut réveiller Lukas ?)

3. Utilisez les constantes, les variables, les conditions et le moniteur série

Bon, J'espère que vous avez fait le plein de sommeil, d'énergie, d'attention, bref que vous êtes à bloc ! Ce qui s'annonce dans ce chapitre va vous titiller un peu les méninges. Assurez-vous donc que vous avez bien compris le chapitre précédent, et en avant pour de nouvelles notions, nouveaux mots de vocabulaire et nouveaux exercices de réflexion !

L'Arduino permet d'aller assez loin dans la programmation. Pour ceux qui développent, la manipulation de données (stockage, transformation, portée) n'est pas différente. En revanche, la communication entre l'Arduino et le PC ou l'Arduino et les montages, apporte quelques nouveautés (nouvelles fonctions, nouvelles constantes...).

Nous aborderons dans ce chapitre la notion de variables et de constantes, indispensable à la programmation. Nous verrons comment les créer, les manipuler, et surtout leur intérêt dans un programme.

Nous aborderons aussi le moniteur série de l'IDE (logiciel de programmation de l'Arduino), pour le moment dans son utilisation d'affichage des données provenant de la carte Arduino. Vous verrez qu'elle est souvent d'une grande utilité.

Enfin nous finirons par la conception d'un programme (codé de deux façons différentes), qui ne nécessite pas de montages électroniques particuliers.

J'en profiterai pour vous apporter la notion de condition qui est d'une utilisation fondamentale dans toute programmation.

Allez, on jette son chewing-gum, on s'installe confortablement et on envoie la purée !

Constantes et variables

Constantes

Dans le chapitre précédent, nous avons construit un programme qui permet de faire clignoter la LED 13. Certains des mots clés étaient colorés de bleu : `HIGH` , `LOW` , `OUTPUT` . Ce sont des constantes. Pour nous, humains, ces trois mots sont intelligibles, mais pour la machine, ils ne veulent rien dire. En revanche, lors de la compilation (voir chapitre précédent) l'IDE transforme ces mots en nombres que le microcontrôleur va comprendre.

Quels sont ces nombres ? Et pourquoi on ne les met pas directement dans le code ?
Dans le cas de`HIGH` et`LOW`, on peut remplacer les mots respectivement par 1 et 0 (faites l'essai) et ça fonctionne. En revanche pour`OUTPUT` je n'ai pas d'idée du nombre correspondant, et vous savez quoi ? On s'en moque. En effet, c'est tout l'intérêt des constantes ! Le mot-clé correspond à un nombre dont nous ignorons tout, mais nous connaissons l'utilisation du mot clé. L'IDE se chargera de la traduction. Pour nous le programme est clair, et pour l'Arduino aussi. Tout le monde est gagnant.
Alors les constantes en bleu (`LOW` , `HIGH` , `OUTPUT`) sont réservées à l'IDE. Nous pouvons les utiliser mais pas les définir. Mais là où ça devient intéressant, c'est qu'on peut se créer ses propres constantes.
Vous avez remarqué que nous avons écrit le nombre 13 trois fois au total dans le programme. Ce nombre dans notre programme n'a pas de raison de changer. Il va être constant pendant toute l'exécution. Nous pourrions donc en faire une constante ! Et nous allons le faire !!

Reprenez le programme et modifiez le comme suit (je modifie les commentaires) :

```
// on définit la constante en donnant un nom et un nombre.

const int CONNEXION=13;



void setup()

{

    // on l'utilise en écrivant juste son nom

    pinMode(CONNEXION,OUTPUT);

}

void loop()

{

    // là encore.

    digitalWrite(CONNEXION,HIGH);

    delay(1000);

    // et là encore.

    digitalWrite(CONNEXION,LOW);

    delay(1000);

}
```

Voici le rendu sur l'IDE :

```
1
2  // on définit la constante en donnant un nom et un nombre.
3  const int CONNEXION=13;
4
5  void setup()
6  {
7      // on l'utilise en écrivant juste son nom
8      pinMode(CONNEXION,OUTPUT);
9  }
10
11 void loop()
12 {
13     // là encore.
14     digitalWrite(CONNEXION,HIGH);
15     delay(1000);
16     // et là encore.
17     digitalWrite(CONNEXION,LOW);
18     delay(1000);
19 }
```

Bon plusieurs points à signaler :

- On écrit le nom des constantes en majuscules (ça fait partie des conventions de programmation) ça permet de savoir que ce sont justement des constantes.
- Le nom de notre constante (CONNEXION) ne s'écrit pas en bleu car elle n'est pas une constante réservée à l'IDE mais bien une à nous, qu'on a fait tout seul comme un grand !
- La déclaration d'une constante (c'est-à-dire sa construction) commence par le mot clé **const** suivi du type de constante (dans notre cas la constante CONNEXION est de type “int”, qui signifie nombre entier - nous y reviendrons plus loin).
- La déclaration d'une constante est toujours de la forme :

```
const type NOMDELACONSTANTE=valeur;
```

Le signe = en informatique n'a pas le même sens qu'en mathématiques ! Dans le cas de la déclaration d'une constante ou d'une variable, il a un rôle d'affectation, c'est-à-dire dans ce cas il dit à l'ordinateur : tu stockes le nombre 'valeur' dans une case de ta mémoire, et tu l'associes au mot 'NOMDELACONSTANTE'.
Alors comment ça marche pour l'IDE ? Et bien dans le cas de constantes, il ne va rien stocker dans la mémoire de l'Arduino. Au moment de la compilation (transformation du programme en langage machine) il va remplacer chaque mot CONNEXION par la valeur 13. C'est la raison pour laquelle une constante n'est pas modifiable, car à aucun moment le microcontrôleur sait qu'on en a utilisé une.

Tenter de modifier une constante en cours de programme crée une erreur de compilation.

Alors je vois à vos yeux sceptiques plissés et interrogateurs que vous vous demandez pourquoi on s'embête à ajouter une ligne de code et remplacer le nombre 13 (2 caractères) par le mot CONNEXION (9 caractères) pour obtenir le même résultat ?

Et bien deux raisons vont me permettre de transformer ces visages fermés en visages auréolés de compréhension...

Tout d'abord, l'utilisation de constantes facilite la lecture du programme : rien ne ressemble plus à 13 que 13 alors que CONNEXION est plus parlant en terme de contexte. Cela nous permet donc de voir plus facilement quand on a utilisé la valeur de la constante ou quand il s'agit d'autre chose.

La seconde raison est le gain de temps. Ici notre programme est minuscule, et nous perdons en effet du temps à changer 13 en CONNEXION. Imaginez maintenant un programme de 300 lignes (ça reste faible ) qui contient 127 fois le nombre 13, dont 56 fois liés à la valeur de la connexion et le reste lié à des valeurs utilisées à d'autres fins. Et là vous vous dites : je préfèrerais utiliser finalement la connexion 7. Vous allez devoir tout remplacer à la main (l'outil de recherche/remplace ne ferait pas le bon travail car il remplacerait tout) et vous en aurez pour un bout de temps et bien des risques d'erreur. Alors qu'avec le principe de définition des constantes, vous n'aurez qu'à modifier une seule ligne : celle de la déclaration de la constante en début de programme, l'IDE se chargera du reste !

On s'arrête là pour le moment en ce qui concerne les constantes. Passons aux variables.

Variables

Alors si les constantes sont constantes, c'est qu'elles ne varient pas pendant l'exécution du programme, donc vous voyez où je veux en venir... ben oui ! Les variables elles, peuvent varier, c'est à dire changer de valeur au cours du programme. C'est hyper-méga pratique !!!

La déclaration de variable peut se faire de différentes façons. Pour des raisons de bonnes habitudes et de lisibilité, nous allons utiliser une méthode en deux temps. Je la trouve plus utile dans le cas où vous voudriez un jour programmer sur autre chose que l'Arduino et en plus elle est moins coûteuse en espace mémoire (ce qui, vous le verrez est important pour les microcontrôleurs).

- *Premier temps* : je dis à l'IDE que je réserve une place en mémoire pour une variable et je lui donne un nom (cette opération est faite une seule fois dans le programme).
- *Autres temps :* je dis à l'IDE d'affecter une valeur à la variable (cette opération peut être faite à tout moment dans le programme).

Comment faire ? Voyez plutôt dans ce bout de code :

```
int nombreDeBilles;//on déclare une varialbe de type int nommée nombreDeBilles

nombreDeBilles=4;//on affecte le nombre 4 à la variable nombreDeBilles

nombreDeBilles=8;//on affecte le nombre 8 à la variable nombreDeBilles
```

Le mot-clé`const` n'est plus là car il s'agit bien de déclarer une variable et pas une constante. Dans le cas de variables, l'IDE ne procède pas comme pour les constantes. Lors de la déclaration, il va traduire en langage machine qu'il faut réserver une place de la bonne dimension (c'est le type) dans la mémoire vive (c'est-à-dire modifiable) de l'Arduino. Puis,

lors des modifications de variables, l'Arduino ira déposer dans sa mémoire, à la place prévue, la nouvelle valeur, qui à chaque fois remplacera la précédente.

Tant qu'aucune valeur n'est attribuée à la variable, la case mémoire contient n'importe quoi. Si vous utilisez une variable non initialisée, vous obtiendrez des résultats souvent étonnants qui risqueront de planter votre programme.
Pour initialiser une variable, on peut utiliser la boucle du`setup()` ou un autre endroit dans le programme. L'important c'est que la variable soit initialisée avant son utilisation.
Voici un exemple de code :

```
int tempsDePause;// déclaration de la variable tempsDePause

void setup()

{

    tempsDePause=100; // initialisation de la valeur de tempsDePause à 100

}



void loop()

{

    // utilisation de la variable tempsDePause dans un delay()

    delay (tempsDePause);

    // changement de la valeur de tempsDePause à 200

    tempsDePause=200;

}
```

Si vous téléversez ce programme, il va tourner et modifier la valeur de la variable`tempsDePause` comme vous lui avez demandé, mais il ne vous affichera rien du tout. Nous verrons plus loin comment faire pour qu'un programme affiche des valeurs à l'écran. Mais avant cela un petit point sur le type des données.

On peut déclarer et initialiser une variable en deux étapes comme ceci :

```
int tempsDePause; // déclaration de la variable

tempsDePause=100; // initialisation
```

... ou bien en une seue étape comme ceci :

```
int tempsDePause=100;
```

La méthode en deux étapes est légèrement moins gourmande en mémoire. Elle n'est nécessaire que si votre programme est très long et gourmand en mémoire...

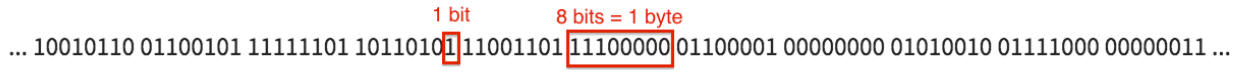
Le typage des variables et constantes

Cette notion est un peu rébarbative, mais c'est comme le dentiste, faut y passer.

Le **typage** est obligatoire lorsqu'on définit une variable ou une constante. Dans certains langages (comme le Python par exemple), il n'est pas nécessaire car le compilateur se chargera de le faire et l'exécution du programme s'occupera des modifications éventuelles. Mais pour l'Arduino (et le langage C) on n'a pas le choix.

Chaque type de variable ne prend pas le même espace en mémoire et ne sera pas traité de façon équivalente lors de l'exécution. Je ne vais pas vous faire un cours pesant sur la mémoire des processeurs (bien des cours sur ce site le font très bien) mais je vais vous en faire un survol.

La mémoire d'un processeur (gros ou petit) est un ensemble d'espaces de même taille qui se suivent (virtuellement sur une même ligne). Et comme ils se suivent on peut les retrouver grâce à leur position sur cette ligne virtuelle. Leur taille est d'un **byte**, c'est-à-dire 8 bits. Le **bit** (non Lukas, je n'ai pas fait d'erreur !) c'est l'unité de base en informatique et ça vaut 0 ou 1. Donc on imagine des 0 et des 1 qui se suivent, formant une ligne qui s'étend sur des kilomètres, par groupes de 8 bits. Genre :



Et bien chaque type de variable (un peu comme les bateaux à la bataille navale) prend un nombre de tranches différent pour être stocké. La taille minimum est de 1 tranche (un byte).Quand on utilise une variable de taille 3 tranches, le processeur va cherche sur la ligne à quelle tranche commence le stockage (c'est **l'adresse**) et va récupérer le contenu des trois tranches qui se suivent à partir de cette adresse et les transformer en valeur. Donc si le typage n'existait pas pour préciser l'espace pris en mémoire par la variable, on récupérerait soit trop soit pas assez d'infos pour la valeur attendue.

Attention à ne pas confondre **byte** et **bit** ! Retenez-bien : 1 **byte** = **8bits.**
Je vous donne quelques types et leur taille pour commencer (il en existe beaucoup plus) ainsi que ce qu'ils peuvent stocker :

type	taille en byte	valeurs stockées
boolean	1	true ou false
char	1	un caractère ou un entier entre -128 et 127
unsigned char	1	un entier entre 0 et 255
byte	1	un entier entre 0 et 255
int	2	un entier entre -32768 et 32767
unsigned int	2	un entier entre 0 et 65535
float	4	un décimal, précis à 7 chiffres après la virgule

"Et, je te le dis, je n'irai pas plus loin." (J.Brel)

Ça nous suffit largement pour le moment ! Vous verrez qu'avec ces types de variables on peut faire bien des choses !

Petit point sur les conventions des noms de variables

Nous l'avons vu pour les constantes, il est de bon ton de tout mettre en MAJUSCULES.

Pour les variables il existe une façon agréable, lisible et surtout conventionnelle de les nommer : on écrit le nom tout attaché et on met des majuscules à chaque mot sauf au premier. Voici un exemple de nom de variable :

```
int valeurDuContactAvantDroit;
```

Au début on a tendance à vouloir mettre des noms de variables courts (genre une lettre) parce qu'on a la flemme d'écrire. Mais très vite on s'aperçoit que des noms de variables clairs (presque des phrases) facilitent la lecture du programme et sa compréhension. Habituez-vous très vite à cette façon de faire, vous gagnerez par la suite beaucoup de temps. D'autant que la taille du nom d'une variable n'a pas d'influence sur le programme.

Pas d'accent, pas d'espace dans les noms de variable. Évitez en général les caractères spéciaux. Interdit aussi de commencer un nom de variable par un chiffre.
Bon, vous savez déclarer des variables, mais pouvez-vous le faire n'importe où dans le programme ?

Portée des variables ou "scope"

Alors... la portée des variables (appelé aussi scope) c'est une façon commode de dire que là où on déclare une variable, c'est là où on va pouvoir l'utiliser et pas ailleurs.

Rappelez-vous que les accolades définissent des blocs de code à exécuter. Et bien si vous définissez une variable dans un bloc de code (donc entre des accolades), elle ne pourra être appelée ou modifiée QUE dans CE bloc de code.

Pour le moment vous ne connaissez que 2 types de blocs de code :

- `setup()` (qui s'exécute une fois), et
- `loop()` (qui s'exécute à l'infini).

Vous verrez qu'il en existe bien d'autres.

Une fois que l'on sort des accolades du bloc, la place utilisée par la variable en mémoire est libérée. Ce qui nous laisse la possibilité de l'utiliser pour une autre variable.

Mais dans les exemples précédents, on a déclaré la variable au tout début, donc ni dans le setup(), ni dans la loop() !
Bien joué Callaghan ! Vous venez de mettre le doigt sur une forme de déclaration qui est très utilisée avec l'Arduino (et très déconseillée dans les autres types de programmation) qu'on appelle la déclaration de **variables globales**.

Une variable globale est accessible (et modifiable) depuis n'importe quel endroit du programme. C'est pratique, mais ça nécessite de faire attention à son nom pour bien la repérer et surtout, une variable globale ne libère jamais sa place occupée en mémoire. L'utilisation excessive de variables globales peut parfois gêner l'exécution du programme en particulier pour l'Arduino dont la place mémoire est très restreinte.

Je crois que vous avez votre compte avec les variables, les constantes et la mémoire pour le moment. Nous allons maintenant nous amuser un peu avec.

Le moniteur série

Bon avant de commencer ce point, je vais me prendre un petit café... je vous en offre un ? (oui Lukas, j'ai aussi du tilleul menthe, mais je ne comprendrai jamais pourquoi vous y trempez votre sandwich rillettes et saumon !)

Grâce à la magie de l'espace temps, me voici déjà de retour et prêt à vous nourrir de savoir !

La console (ou moniteur série) n'est pas faite pour faire tourner des jeux comme GOW, GTA, AC ou FIFA, n'en déplaise à certains (Lukas puisque je vous regarde, il vous reste un morceau de saumon sous la narine droite). Non, la console, même si on peut s'amuser avec, est un moyen simple d'afficher des infos provenant de votre Arduino.

Comme vous l'avez sûrement remarqué, l'Arduino de base n'a pas d'écran. On peut lui en ajouter un, nous aurons l'occasion de le voir plus tard, mais en attendant, on ne peut rien afficher. C'est là qu'entre en scène le **moniteur série**, qui est une fenêtre que l'on peut ouvrir en cliquant sur la loupe du logiciel Arduino. Cette fenêtre reçoit et envoie des infos sous forme de séries de bytes aux ports concernés. Donc grâce au moniteur série, l'Arduino peut (à condition d'être connecté à un PC) envoyer des infos à l'ordinateur qui va pouvoir les afficher en temps réel.

Pour ce faire on fait appel à une bibliothèque spéciale, déjà incluse dans l'IDE : la bibliothèque **Serial.**

Une **bibliothèque** est un fichier dans lequel sont stockés des sous-programmes, des variables et des constantes. Une fois ajouté dans notre programme, on a le droit d'utiliser leur fonctionnalités et leurs types de données. On utilise pour ce faire les mots-clés liés à la **bibliothèque** utilisée.
Pour utiliser la possibilité de communication entre l'ordinateur et l'Arduino, on procède en deux étapes :

1. On initialise la communication (ça se fait dans le`setup()`).
2. On communique (ça se fait souvent dans la`loop()` mais possible dans le`setup()`).

Dans le programme suivant, vous allez découvrir de nouveaux mots-clés de communication comme`Serial.begin`, `Serial.println`... Ne vous inquiétez pas, on y arrive ! Pour l'instant, contentez-vous de lire les commentaires qui vous permettront de comprendre le principal.

```
void setup()

{

    Serial.begin(9600);// initialisation de la communication

    Serial.println("Communication initialisée");// envoi d'un message

}

void loop()
```

```
{

    Serial.println("Je suis dans la boucle !");//envoi d'un autre message

}
```

Écrivez-le, téléversez-le. Si tout va bien, rien ne se passe, sauf que la diode TX de votre Arduino est allumée.

La LED TX montre une transmission envoyée par l'Arduino, la LED RX montre une transmission reçue par l'Arduino.
Pour comprendre le principe d'utilisation d'une bibliothèque :

- Les codes liés à une bibliothèque commencent tous par son nom (ici Serial) suivi d'un point et de la commande.
- La commande est spécifique à la bibliothèque utilisée.
- La liste des commandes liées à une bibliothèque est ce qu'on nomme la documentation. Elle est souvent liée à la bibliothèque.
- Pour la bibliothèque Serial, la documentation et les exemples se trouvent sur le site officiel de l'Arduino, dans le menu learning/reference. Je vous donne le lien direct.

Bien, maintenant, cliquez sur le bouton d'affichage du moniteur.

Une fenêtre s'ouvre et là c'est la folie ! Ça défile vers le bas, on a l'impression de rien pouvoir lire, on se demande comment ça s'arrête et si ça ne va pas exploser !

Non, en fait tout est normal. La diode TX est allumée (en fait elle clignote très vite) car elle envoie les messages de l'Arduino vers l'ordinateur. L'IDE les reçoit et les affiche les uns à la suite des autres, sans arrêt. C'est clair que la vitesse d'écriture ferait tomber dans les pommes un moine copiste du moyen-âge. Et encore c'est assez lent car on a le temps de le voir.

Le 9600 dans les parenthèse de l'initialisation correspond à un nombre de caractères par seconde qu'on appelle des bauds. L'Arduino peut donc envoyer un maximum de 9600 caractères par seconde à l'ordinateur dans cette configuration. Ne pas confondre avec l'unité bps, qui veut dire bits par seconde. Un caractère pour l'Arduino c'est 8 bits.
Que diriez-vous d'un petit exercice ?
Je vous propose de modifier ce programme pour que le message dans la boucle ne soit envoyé que toutes les 2 secondes.

Si vous avez réussi, vous remarquerez que la diode TX ne clignote qu'à l'envoi du message.

Vérifiez la solution !

Avant de nous lancer dans des exercices plus conséquents, vous allez avoir besoin de savoir certaines choses sur le moniteur série :

- Le moniteur série utilise les connexions 0 et 1 de l'Arduino (0 pour la diode RX et 1 pour la diode TX lors d'une utilisation en communication). Donc si on utilise le port série, il ne faut pas utiliser les connexions 0 et 1 pour d'autres choses dans votre projet.

- L'affichage des caractères accentués ou spéciaux n'est pas pris en compte (vous l'avez peut-être remarqué lors de l'affichage du message "communication initialisée" qui donne un résultat bizarre), il faudra donc vous en passer.
- La vitesse de communication entre Arduino et l'ordinateur peut prendre d'autres valeurs que 9600 bauds. Vous trouverez ces valeurs sur le site officiel avec ce lien. 9600 est la valeur généralement utilisée.
- Si vous appuyez sur le bouton reset de l'Arduino, cela n'effacera pas les messages précédents affichés sur le moniteur. Pour lancer le moniteur, ça ne se fait pas automatiquement : il vous faut appuyer sur le symbole de la loupe que l'on a vu plus haut. Une fois lancé, le programme de l'Arduino recommencera alors du début.

À partir de maintenant, nous appellerons **pin**, les connexions disponibles sur l'Arduino. Le pin 13 correspondra donc à la connexion 13 (ou broche 13, ou trou 13).
(Je suis surpris Lukas de ne pas vous entendre sortir une de vos blagues habituelles...)

Bon, vous êtes fins prêts pour quelques exercices à base de moniteur, de constantes et de variables !

Programme "Hello Arduino World !"

Ceux qui ont déjà appris un langage de programmation savent que l'exemple de base, pour chaque langage, consiste à afficher un "bonjour au monde" (Je me suis toujours demandé d'ailleurs si ce n'est pas une thérapie pour les développeurs asociaux que nous sommes plus ou moins, afin de les rassurer sur le fait qu'il ne sont pas seuls). Nous n'échapperons (rouge) pas à la règle et nous allons nous aussi réaliser ce programme, enfin c'est plutôt vous qui allez réaliser ce programme.

Ce que vous devez obtenir : à l'ouverture du moniteur, un message qui affiche sur trois lignes "Hello Arduino World !" avec un mot par ligne (le point d'exclamation avec le dernier mot). **Le message ne doit pas se répéter à l'infini.**

Commencez en cliquant sur "nouveau" (3ème bouton). L'étendue de vos connaissances vous permet de réaliser seul(e) ce programme. Il va peut-être vous falloir faire deux ou trois tests et quelques débuggages (néologisme qui est assez clair). Cet exercice n'a d'intérêt que si vous arrivez au bout qu'à la force seule de votre intelligence et vos connaissances liées.

Allez au boulot !

Correction :

```
//Programme hello world

void setup()

{

    //initialisation de la communication

    Serial.begin(9600);

    //affichage du texte
```

```
    Serial.println("Hello");

    Serial.println("Arduino");

    Serial.println("World !");

}

void loop()

{



}
```

Vous remarquerez que l'instruction`Serial.println()` affiche le texte entre guillemets et retourne à la ligne. Modifiez le programme en utilisant l'instruction`Serial.print()` donc en enlevant 'ln' à la fin. Vous verrez que les mots se collent les uns aux autres. On retient donc que :

- `Serial.println()` : affiche avec retour à la ligne
- `Serial.print()` : affiche sans retour à la ligne

Bien, mais le vrai objectif de mon exercice est dans ce qui suit. Je veux obtenir le même résultat, mais en mettant les instructions d'affichages dans la`loop()`. Pour pouvoir le réaliser, il vous faut apprendre un nouveau concept : **la condition**.

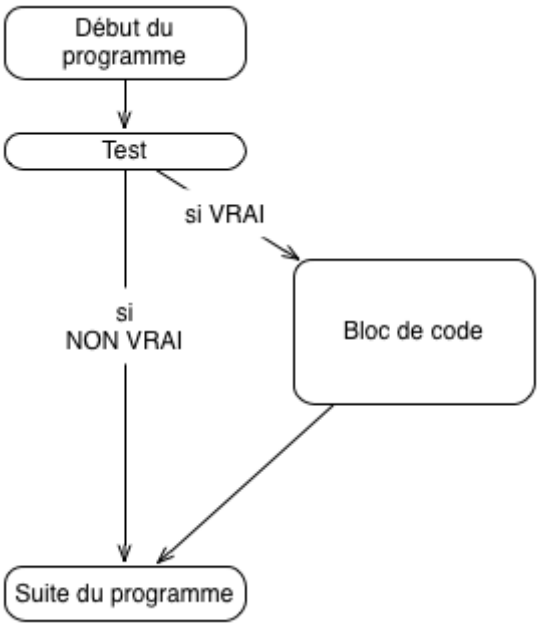
La condition

La condition est un test que va réaliser le programme en se posant une question. S'il peut y répondre 'vrai' alors il exécute un bloc d'action (qui sera mis entre accolades), sinon il ne l'exécute pas.

voici le pseudo code correspondant :

```
debut du programme

si (test)

{

    ce code s'exécute si on peut répondre vrai au test

}

le programme continue
```

Et voici une représentation graphique :



Dans notre programme “Hello Arduino world!”, nous allons ajouter un test pour vérifier si le message a bien été affiché sur la console une fois. Si oui, on ne l'affiche plus.

Voici donc le code correspondant :

```
int affichageFait;// déclaration de la variable



void setup()

{

    Serial.begin(9600);

    affichageFait=0;//initialisation de la variable

}



void loop()

{

    if (affichageFait==0)

    {

        //ce code n'est exécuté que si la condition est vérifiée
```

```
        Serial.println("Hello");

        Serial.println("Arduino");

        Serial.println("World !");

        affichageFait=1;//on passe la variable à 1 pour ne plus exécuter le code

    }

}
```

Une condition s'écrit donc avec le mot-clé **if,** puis la condition à tester est mise entre parenthèses, puis le code à exécuter entre accolades :

```
if (condition) { code à exécuter }
```

Vous remarquerez que pour tester une valeur dans une condition on met un double égal : ==

C'est la façon d'écrire l'égalité dans une condition voici le tableau des signes de condition :

Code	Condition testée
==	égal
>=	supérieur ou égal
<=	inférieur ou égal
>	supérieur
<	inférieur
!=	différent (non égal)

Une erreur de débutant est de confondre le = qui permet d'assigner une valeur à une variable ou une constante, et le == qui permet de tester l'égalité.
Observez bien le code précédent pour bien intégrer le mécanisme de condition. Vous verrez par la suite qu'on le complétera et qu'on l'utilisera dans bien des cas. À la fin de cette première partie du cours, vous serez bien plus à l'aise avec le principe.

En résumé

Vous avez rencontré dans ce chapitre beaucoup de nouveauté :

- Les constantes qui remplacent des valeurs et ne peuvent pas être modifiées en cours de programme
- Les variables qui stockent des données et peuvent être modifiées en cours de programme
- Le typage qui définit la taille prise par les variables ou les constantes en mémoire
- Le scope qui permet de comprendre où utiliser les variables dans un programme
- La console qui affiche des informations envoyées par l'Arduino

- La bibliothèque Serial, qui facilite les affichages sur la console
- La condition `if (condition){code}` qui permet d'exécuter un code en fonction d'une condition.

Bon c'était dense et assez compliqué sur la fin. Mais vous allez pouvoir vous poser un peu au prochain chapitre pour concevoir deux programmes qui vont vous permettre de mieux asseoir vos connaissances. Il vous révélera aussi un dernier concept avec les boucles, et vous aurez les bases suffisantes pour vous concentrer sur l'électronique dans les chapitres suivants.

4. Faites des boucles et des calculs

Dans tout apprentissage, la pratique reste souvent un bon moyen de mieux comprendre la théorie.

Ce chapitre va tourner autour de la réalisation de deux programmes qui devraient vous permettre d'appliquer les connaissances que vous venez d'acquérir dans ces premiers chapitres :

1. Le premier programme va vous permettre de vous amuser avec les variables en laissant le programme les modifier à votre place. Nous en profiterons pour réviser la notion de condition et aborder la notion de boucle qui permet bien des choses en programmation.
2. Le second devrait faire appel à tout ce que vous avez vu dans ces premiers chapitres, et aura pour objectif de programmer des affichages sur le moniteur.

Programme "Arduino compte seul"

L'objectif de ce programme est de faire compter l'Arduino tout seul. Je m'explique :

- Tout d'abord, l'Arduino doit afficher sur le moniteur série le nombre 1 et allumer une fois la LED 13.
- Puis, après un temps d'arrêt d'une seconde, il doit afficher le nombre 2 et faire clignoter 2 fois la LED 13 (à 2Hz, je vous laisse revoir le cours sur les fréquences).
- Encore un arrêt d'une seconde, puis affichage du chiffre 3 avec 3 clignotements (toujours à 2 HZ).
- L'opération se répète jusqu'au nombre 20, puis l'Arduino envoie "Ayé !" sur le moniteur série et plus rien ne se passe.

Pour réaliser ce programme, nous allons procéder en deux temps. Mais avant de nous lancer, laissez-moi vous présenter quelques notions dont vous allez avoir besoin.

Calculs sur variable

Tout d'abord, il vous faut savoir qu'une variable peut être modifiée par calcul. Jusque là, nous avions écrit :

```
int uneVariable; //uneVariable contient on ne sait trop quoi

uneVariable=10; //uneVariable contient 10
```

```
uneVariable=32; //uneVariable contient 32
```
Et bien il est possible d'écrire :

```
int uneVariable; //uneVariable contient on ne sait trop quoi

uneVariable=10; //uneVariable contient 10

uneVariable=10+22; //uneVariable contient 32 (résultat de 10+22)
```
Mais là où ça devient intéressant (si si, je vous assure), c'est qu'on peut écrire ceci :

```
int uneVariable;

uneVariable=10;

uneVariable=uneVariable+22;
```


Comment ça marche ? Tout simplement comme votre cerveau si je vous dis :

"Imaginez un bus. Dans ce bus, il y a 10 personnes. Il en monte 22, combien le bus contient de passagers ?"

Voilà ce que vous faites et ce que fait l'ordinateur :

Phrase	Action humaine	Action machine	Code
Imaginez un bus	vous allez d'abord imaginer le bus	déclaration de la variable bus	int bus;
Dans ce bus il y a 10 personnes	vous allez y mettre 10 personnes	initialisation de la variable à 10.	bus=10;
Il en monte 22	vous allez reprendre le nombre de personnes	appel de la variable bus	bus
Il en monte 22 (suite)	vous lui ajoutez 22	calcul sur la variable bus	bus+22
Il en monte 22 (fin)	vous mettez tout le monde dans le bus	affectation du résultat dans la variable bus	bus=bus+22;

N'oubliez pas que le signe= est un signe d'affectation ! Il faut imaginer`bus=bus+22` comme`bus+22=>bus` (`bus+22` va dans`bus`). On modifie la valeur de`bus` par le résultat de `bus+22`
Si vous avez compris cela, la suite ne devrait pas vous poser de problèmes. Voici ce qui est encore possible :

```
//Déclarations

int a;
```

```
int b;

int c;



//Initialisations

a=3; // a vaut 3

b=5; // b vaut 5

c=0; // c vaut 0



//Calculs

c=b+a; // c vaut maintenant 8

a=a+c; // a vaut maintenant 11 car 3+8

b=a+b; // b vaut maintenant 16 car 11+5

c=a+b; // c vaut maintenant 27 car 11+16
```

Prenez un instant pour être sûr(e) de bien avoir saisi ce code. Faites les calculs de tête, vérifiez.

Nous allons faire un petit programme sur l'Arduino qui compte de 1 en 1 et qui nous l'affiche sur le moniteur. Vous pouvez essayer de le réaliser seul(e) avant de regarder le code.

```
int compteur; //déclaration d'une variable compteur



void setup()

{

    Serial.begin(9600); //initialisation communication

    compteur=1; // initialisation de compteur

}



void loop()
```

```
{

    Serial.println(compteur); //affiche la valeur de compteur

    compteur=compteur+1; //on ajoute 1 à compteur

}
```

Saisissez le code, téléversez-le et ouvrez le moniteur. Si tout se passe bien, vous voyez l'affichage qui progresse de 1 en 1.

Vous remarquerez que pour afficher une variable on écrit juste le nom de la variable entre les parenthèses de`Serial.println()`, sans guillemets. En fait, les guillemets permettent d'afficher du texte, sinon c'est une valeur qui s'affiche.
Devinette : Si vous laissez tourner ce programme, Arduino n'atteindra jamais 33000. Essayez de trouver pourquoi... (indice : c'est en rapport avec le type de la variable)

Le fait d'ajouter 1 à une variable s'appelle **incrémenter**, et le fait d'enlever 1 c'est **décrémenter**. Ce sont des mots de vocabulaire à connaître.

Bon, si nous reprenions notre objectif de programme de début de chapitre, nous pourrions dire que faire compter l'Arduino jusqu'à 20 va être finalement assez simple. Il faut juste savoir comment l'arrêter... quelqu'un a une idée ?

(non Lukas, le débrancher n'est pas la réponse que j'attendais...)

Et bien nous allons utiliser une condition ! Nous allons dire à Arduino : tant que le compteur est inférieur ou égal à 20, tu comptes, sinon tu ne fais rien. Ce qui donne en pseudo-code :

```
déclaration de la variable compteur



void setup()

{

    initialisation de la communication

    initialisation de compteur à 1

}



void loop()

{

    si (compteur est inférieur ou égal à 20)
```

```
    {

        affichage de la valeur de compteur sur le moniteur

        incrémentation de compteur

    }

}
```

Je vous laisse le traduire en vrai code, c'est un bon exercice.

L'utilisation de pseudo-code est assez efficace pour organiser son programme. Il permet aussi de présenter ce qu'on attend en s'affranchissant des mots-clés précis du langage. Un développeur de C pourrait sans problème le traduire pour un affichage sur la console.

Allez, voici le code pour l'Arduino :

```
int compteur;

void setup()

{

    Serial.begin(9600);

    compteur=1;

}

void loop()

{

    if (compteur<=20)

    {

        Serial.println(compteur);

        compteur=compteur+1;

    }

}
```

Comme nous l'avons vu au chapitre précédent, si vous appuyez sur le bouton d'affichage du moniteur (la loupe), ça relance le programme sur l'Arduino. Mais une partie des envois précédents a eu le temps de passer et de s'afficher. Tout cela manque parfois de clarté. On peut donc ajouter une ligne de présentation juste après l'initialisation de la communication :

```
Serial.println("*** Debut du programme ***");
```

Ok, mais maintenant, comment on peut faire que la LED 13 clignote le nombre de fois spécifié dans`compteur` ?
Ha ! Ça fait plaisir de voir que ce cours vous maintient en haleine (non merci pas trop près Lukas) et vous pousse ainsi vers le savoir !

Et bien pour répondre à cette excellente question, je vais en profiter pour aborder une notion particulièrement utile : les boucles !

La boucle "for"

La programmation nous permet de réaliser des boucles dans un programme. C'est à dire qu'on peut exécuter un bloc de code plusieurs fois avant de passer à la suite. Cette répétition ouvre un grand nombre de possibilités.

Il existe plusieurs sortes de boucles, nous les verrons progressivement. Nous allons commencer par la boucle`for`. Elle permet de répéter un code un nombre de fois connu à l'avance.
Elle se présente un peu comme une condition : elle commence par un mot-clé (`for`), suivi de certaines choses entre parenthèses, puis des accolades qui contiennent le code à exécuter.
Voici un exemple :

```
for (int t=0;t<10;t=t+1)

{

    code à exécuter

}
```

Le plus difficile à comprendre se situe entre les parenthèses. Il y a trois informations importantes séparées par des points-virgules :

1. **Initialisation** de la variable qui va servir de compteur. Dans notre exemple, on déclare et on initialise (dans le même temps) une variable t de type "int" :`int t=0` .
2. **Condition à tester** pour continuer ou non la répétition. Ici, on exécute le code tant que t est inférieur à 10 : `t<10`.
3. **Action sur le compteur** à chaque tour de boucle. Dans notre cas, t est augmenté de 1 à chaque passage : `t=t+1`.

Ce qui est encore plus fort, c'est que cette variable t (le compteur) peut être utilisée dans le code à exécuter ! Du coup, voici notre programme de comptage modifié avec une boucle for. Observez-le, testez-le et comprenez-le !

```
void setup()

{

    Serial.begin(9600);
```

```
    Serial.println("*** Debut du programme ***");

}

void loop()

{

    for (int compteur=0;compteur<=20;compteur=compteur+1)

    {

        Serial.println(compteur);

    }

}
```

Il compte bien jusqu'à 20, mais reprend ce comptage sans arrêt !

C'est normal, nous n'avons pas indiqué à Arduino qu'il ne doit le faire qu'une seule fois. Il va donc falloir creér une variable à tester pour le lui indiquer.

Le type boolean

Je vais donc en profiter pour vous parler des variables de type **boolean** (ou booléens). Ce sont des variables qui ne peuvent prendre que deux valeurs : soit 1 (true, c'est-à-dire vrai), soit 0 (false, c'est-à-dire faux). Les booléens sont des variables pratiques pour faire des tests comme celui dont nous avons besoin pour vérifier si le compteur de notre programme “Arduino compte seul” a fini de compter jusqu’à 20.

On peut écrire`boolean a=1;` ou`boolean a=true;` la valeur de true est 1.
Observez le bout de code suivant :

```
boolean etat; //déclaration de la variable etat de type boolean

etat=true; //initialisation de etat à VRAI

if (etat) // test si etat est VRAI

{

    //bloc de code exécuté si etat est VRAI

}
```

Vous remarquerez que l’on n’a pas besoin d’écrire`if(etat==true)` dans la condition, il suffit d’écrire`if(etat)` qui signifie la même chose (mais les deux sont corrects).
Il faut comprendre que si vous créez une variable booléenne “drapeau” avec l’instruction suivante :

```
boolean drapeau;
```

Les trois instructions suivantes lui assigneront la même valeur “true” :

```
drapeau=true;   //drapeau vaut true

drapeau=1;      //drapeau vaut true, car true c'est 1

drapeau= 3<5;    //drapeau vaut true, car 3 est inférieur à 5
```

Testez ce petit programme sur l'Arduino :

```
boolean etat;



void setup()

{

    Serial.begin(9600);

    etat=true;

}

void loop()

{

    etat=3<5;

    Serial.println(etat);

}
```

Il affiche des 1. Car`etat` vaut 1(true) résultat du test (3<5).
Allez, on modifie notre programme "Arduino compte seul" pour y ajouter notre test de fin :

```
boolean affichage; //variable pour stopper l'affichage

void setup()

{

    Serial.begin(9600);

    affichage=true;  //initialisation de la variable à true

    Serial.println("*** Debut du programme ***");
```

```
}

void loop()

{

    if (affichage) // test si affichage vaut true

    {

        //boucle de comptage

        for (int compteur=0;compteur<=20;compteur=compteur+1)

        {

            Serial.println(compteur);

        }

        affichage=false; // on passe affichage à false

    }

}
```

Vous voyez ici un bel entrelacement d'accolades. Pour s'y retrouver, il est conseillé, à chaque nouveau niveau d'accolade de faire une tabulation. Ainsi c'est plus lisible. Vous avez un outil de formatage automatique dans le menu Outils de l'IDE. C'est très pratique, essayez-le.

De plus, l'IDE vous propose un repérage simplifié des paires d'accolades ou de parenthèses. En effet, si vous sélectionnez ou cliquez à la droite d'une parenthèse ou d'une accolade, vous verrez que sa compagne s'encadre en bleu.

Et maintenant, le clignotement ! Rappelez-vous, si le compteur vaut 1, on clignote 1 fois, s'il vaut 2, 2 fois, etc. Quand on regarde le programme, on se dit que ce serait bien que le code de clignotement soit juste après l'affichage de compteur (donc après la ligne 15 dans le code ci-dessus). Comment faire ? Là encore c'est la boucle`for` qui va nous servir. Voici un code pour faire clignoter 10 fois la LED 13 (affiché en coloration syntaxique du langage C) :

```
int nombreDeFois; // variable pour le nombre de fois

int numeroPin; // variable pour le pin utilisé

boolean faireCode; // variable pour arrêter le clignotement
```

```
void setup()
{
    numeroPin=13; // initialisation pin à 13
    pinMode(numeroPin,OUTPUT);
    nombreDeFois=10; // initialisation nb de fois à 10
    faireCode=true; // initialisation à true
}

void loop()
{
    if (faireCode) // test si true
    {
        for (int t=0;t<nombreDeFois;t=t+1) // boucle clignotement
        {
            //allume
            digitalWrite(numeroPin,HIGH);
            delay (250);
            //éteint
            digitalWrite(numeroPin,LOW);
            delay (250);
        }
        faireCode=false;
    }
```

```
}
```

Ce code ne devrait pas vous poser de problèmes si vous lisez les commentaires. J'utilise la même structure que le code précédent, mais sans l'affichage sur le moniteur série.

J'utilise la coloration syntaxique du langage C (ce qui est mieux pour repérer commentaires et code). J'utiliserai dorénavant cette coloration, la coloration de l'IDE de l'Arduino n'étant pas encore disponible.
Le problème est qu'avec ce code, la LED13 clignotera 10 fois à chaque tour (cf ligne 9). Il faudrait pouvoir faire varier cette valeur. Et bien nous utiliserons en fait la valeur de compteur (qui va de 0 à 20) dans la boucle`for` du clignotement !
Je vous livre le code final car il est intéressant à étudier pour bien appréhender les variables, les conditions et les boucles. Observez-le, dépecez-le, et surtout, essayez de le refaire seul(e)s.

Code final du programme "Arduino compte seul"

Voici le code avec les commentaires :

```
boolean affichage; //variable pour stopper l'affichage

int numPin;



void setup()

{

    numPin=13;

    pinMode(numPin,OUTPUT);

    Serial.begin(9600);

    affichage=true; //initialisation de la variable à true

    Serial.println("*** Debut du programme ***");

}



void loop()

{

    if (affichage) // test si affichage vaut true

    {
```

```
        //boucle de comptage
        //compteur s'augmente de 1 à chaque tour
        for (int compteur=1;compteur<=20;compteur=compteur+1)
        {
            Serial.println(compteur);


            //boucle de clignotement
            //compteur sert de limite à la boucle
            //donc le nombre de clignotements augmente à chaque tour
            for (int nbClignote=0;nbClignote<compteur;nbClignote=nbClignote+1)
            {
                //allume
                digitalWrite(numPin,HIGH);
                delay(250);
                //eteind
                digitalWrite(numPin,LOW);
                delay(250);
            }
            delay(1000); //attente de 1s
        }
        affichage=false; // on passe affichage à false
        Serial.println("*** Ayé ! ***");
    }
}
```

Notez que l'on pourrait écrire ce code de manière plus efficace, mais il faudrait faire recours à des notions que vous n'avez pas encore vues donc pour l'instant on va se contenter de cette version.

N'oubliez pas d'ouvrir le moniteur, et amusez-vous à modifier les`delay`, les limites des boucles, etc.
Essayez de comprendre pourquoi dans les boucles`for`, la variable`compteur` est initialisée à 1, alors qu'on initialise`c` à 0.
Pour incrémenter une variable il existe trois codes possibles qui sont tous corrects :

```
int variable;




// pour incrémenter

variable=variable+1; // celui-ci vous connaissez

variable+=1; // autre façon, plus courte

variable ++; // autre façon encore plus courte




// pour décrémenter

variable=variable-1;

variable-=1;

variable --;
```

Petit exercice : Essayez de simplifier le code de "Arduino compte seul" sans que le résultat ne soit changé (aide : il faut enlever ou modifier certaines boucles et tests).

Vérifiez la solution !

Nous avons réalisé ensemble pas à pas ce programme de clignotement, je vous propose maintenant d'en faire un tous seuls comme des grands !

TP : Programme "Table de multiplication"

Dans ce programme, je vais vous demander d'afficher la table de multiplication par 7 des nombres de 0 à 14. Voici quelques règles à respecter :

- Le programme devra afficher la table de multiplication une seule fois, il faudra donc utiliser une condition pour vérifier si elle a été affichée.
- Pour rendre le programme facile à modifier, vous stockerez le nombre 7 dans une variable de type int. Nous apprendrons plus tard comment envoyer une information à l'Arduino depuis le moniteur, ce qui nous permettra d'afficher la table de notre choix.)
- Le moniteur devra afficher une table qui ressemble à ça :

```
************************
Table de multiplication
La table de : 7

0 x 7 = 0
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
11 x 7 = 77
12 x 7 = 84
13 x 7 = 91
14 x 7 = 98

************************
```

Quelques pistes utiles

- Pour afficher un saut de ligne, il suffit d'utiliser l'instruction suivante : `Serial.println()`, sans rien entre les parenthèses.
- Pour faire un calcul de multiplication dans un programme, on utilise l'étoile *. Donc "3x4" en humain s'écrit "3*4" en machine. Et la division s'écrit avec le slash / en machine.

Allez, je vous laisse travailler, je vais faire une sieste. Ne désespérez pas au premier problème rencontré, relisez le cours, essayez, modifiez. Il n'y a pas de pièges dans la conception de ce programme.

Code pour le programme "Table de multiplication"

Voici le code attendu, vous devriez vous y retrouver facilement car tous les mots vous sont maintenant familiers :

```
int numTable; // variable pour la table concernée

boolean affiche; // variable d'affichage



void setup() {

  affiche = true; // initialisation à true

  numTable = 7; // iniialisation à 7

  Serial.begin(9600); //initialisation de l'affichage



  //Affichage de l'entête du programme

  Serial.println("**********************");

  Serial.println("Table de multiplication");

  Serial.print("La table de : "); //pas de retour à la ligne

  Serial.println(numTable); // affichage de la variable

  Serial.println(); // saut de ligne pour aérer

}



void loop() {

  if (affiche) // test si vrai

  {

    // boucle de progression pour la multiplication

    for (int t = 0; t < 15; t++)

    {
```

```
      int resultat = numTable * t; // variable pour stocker le résultat



      // Affichage de la ligne

      Serial.print(t);

      Serial.print(" x ");

      Serial.print(numTable);

      Serial.print(" = ");

      Serial.println(resultat);

    }

    Serial.println(); // saut de ligne

    Serial.println("**********************");

    affiche=false; // passage à false pour ne plus afficher

  }



}
```

Quelques remarques :

- Ligne 21 : j'utilise la méthode d'incrémentation simplifiée **t++,** c'est identique à t=t+1.
- ligne 23 : la déclaration, l'initialisation et l'affectation de la variable est faite en une seule ligne. C'est tout à fait correct, légèrement plus gourmand en place mémoire.
- La variable "resultat" créée en ligne 23 et affichée en ligne 30, n'est pas une obligation. Nous aurions très bien pu écrire en ligne 30 directement : Serial.println(numTable*t);

Bon c'était là encore une bonne partie (de rigolade ?) bien chargée de connaissances.

Dans l'état actuel de votre savoir programmistique, si vous avez réussi à réaliser seul(e)s ce dernier programme, voire celui d'avant, on peut dire que vous êtes fichtrement avancés maintenant en programmation.

Vous êtes donc largement prêt(e)s à passer à l'étape suivante : les connaissances électriques des montages.

5. Jeux de lumière avec une LED et la breadboard

La quantité d'informations ingurgitée dans les chapitres précédents vous a fait prendre un poids cognitif important qui vous permet maintenant de rester stable dans un environnement envahi par les vents de la nouveauté et de l'interrogation.

Et là, des vents, il va y en avoir ! (Attention à la blague de trop Lukas...)

Vous avez vu jusqu'à présent le cerveau d'Arduino, vous allez maintenant étudier son corps !

Au programme de ce chapitre :

- Nous allons commencer par quelques notions d'électricité, car la théorie est importante. Non pas pour faire de vous des professionnel(le)s de ce domaine, mais plutôt pour éviter le : "Oups, j'ai tout grillé..."
- Nous étudierons ensuite cette merveilleuse petite chose lumineuse qu'est la LED, car vous verrez que son utilisation dans les montages est à la fois esthétique et utile.
- Puis nous ferons un point important sur la "breadBoard" qui nous donnera du pain sur la planche et qui nous permettra l'interfaçage avec l'Arduino.
- Enfin forts de ces connaissances nouvelles, nous reprendrons notre programme précédent pour l'adapter à une LED connectée.

Je vous prierai donc de vous serrer la ceinture, et de la boucler ! (J.HIGELIN)

La tension

Il vous faut dès maintenant prendre conscience d'une chose importante : votre Arduino est parcouru d'électricité. Ça paraît pas très grave vu comme ça, mais en fait, ça complique un peu les choses.

Qu'est-ce que c'est que l'électricité ?

D'abord, ça ne se voit pas. On peut seulement en voir les résultats. L'électricité est un déplacement de particules chargées dans un conducteur (rien à voir avec le code de la route), c'est-à-dire que c'est le mouvement de toutes petites choses (invisibles) dans la matière, qui créé le courant électrique.

La charge électrique est une propriété de la matière.
Parmi les charges qui créent le courant électrique, il existe des charges positives notées + et des charges négatives notées -. Deux charges identiques se repoussent et deux charges contraires s'attirent.

Donc l'électricité est un déplacement de particules, c'est un courant. Je vais donc utiliser une analogie (non Lukas pas une nana jolie), celle de la rivière, qui permet de mieux comprendre certains principes électriques. Attention, comme tout modèle, l'analogie a des limites et n'est plus valable dans certains cas.

Imaginez une rivière qui coule de sa source à son embouchure, puis à l'embouchure, on met une pompe qui ramène toute l'eau à la source. Nous aurions un circuit fermé, avec d'un côté un écoulement, de l'autre un pompage.

La première notion électrique à comprendre est la **différence de potentiel** qu'on appelle aussi **la tension électrique** (ce qui n'est pas toujours juste). C'est cette différence de potentiel

qui crée le courant électrique. Pour la rivière, elle représente la hauteur de la source par rapport à l'embouchure. Donc la pente ou le dénivelé. On comprend facilement que si la source est plus haute que l'embouchure, l'eau va se mettre à couler et va créer un courant.

C'est pareil en électricité, si la différence de potentiel électrique existe (c'est à dire la différence entre charges + et -), le courant électrique va circuler (les particules vont être attirées vers la charge inverse).

La tension, ou potentiel électrique se mesure en **Volts** (symbole **V**).

Pour vous donner quelques idées de mesure de tensions de la vie courante :

Exemple	Tension
Pile	1,5 V ; 4,5 V ; 9 V
Batterie voiture	12 V ; 24 V
Prise de courant	110 V ; 220 V
Ligne à haute tension	10 000 V ; 500 000 V
Foudre	1 000 000 V

Le passage du courant électrique dans le corps humain n'est pas anodin, au-delà d'un certain seuil, il provoque des brûlures, tétanise et peut être mortel. Les connexions vers ou depuis l'Arduino doivent respecter les règles de sécurité. Brancher une sortie ou une entrée de l'Arduino directement sur une prise de courant, non seulement grillera votre carte, mais risque de créer un court-circuit ou bien plus.
L'Arduino UNO propose à ses bornes (pin 0 à 13) une tension de +5 V. C'est un courant sans danger (nous verrons la notion d'ampérage plus loin) pour l'homme.

Il utilise un **courant continu**, c'est-à-dire que c'est toujours du +5 V qui sort. Il existe des courants alternatifs qui fournissent une succession de courant positif puis de courant négatif à une fréquence donnée. Par exemple, vos prises de courant fournissent du 220V alternatif à 50Hz. C'est-à-dire que 50 fois par seconde le courant passe de +220V à -220V. (Je dis courant pour simplifier, mais il s'agit bien de tensions).

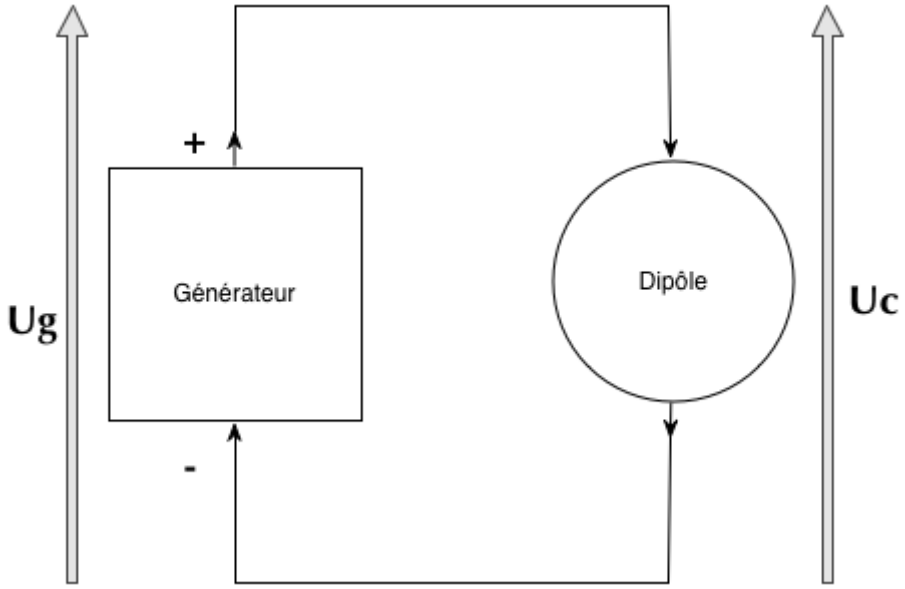
Mais, il y a une connexion pour brancher l'Arduino directement au courant, non ?
NON ! On peut fournir à l'Arduino son alimentation électrique de plusieurs façons, en voici trois principales :

- Le connecter avec le **câble USB** à l'ordinateur. C'est sans danger car l'ordinateur fournit un courant de +5 V par ce câble.
- Connecter une **pile 9 V** (entre 7 V et 12 V en fait) à la prise prévue à cet effet (prise ronde). Attention car il faut vérifier comment brancher le + et le - de la prise sur la pile 9V. Arduino transforme ensuite les 9V en 5V.
- Connecter un **transformateur** à la prise prévue à cet effet (prise ronde). Attention, il faut être sûr que la connectique du transformateur soit correcte et qu'il fournisse bien la tension demandée (9 V). Une mauvaise connexion peut endommager définitivement votre Arduino.

C'est vrai que le transformateur se branche au 220V, mais à l'intérieur du boîtier, il y a deux bobines qui permettent de transformer ces 220V en 9V. Puis un redresseur de courant qui permet de transformer l'alternatif en courant continu.
Représentation d'un circuit électrique

Lorsque l'Arduino est branché à une source d'alimentation électrique, il fait donc partie d'un circuit électrique fermé. Jetons un œil à la structure d'un circuit électrique et ses règles importantes.

Voici le schéma d'un circuit électrique simple. Un dipôle est un composant électrique qui a deux bornes pour connecter des fils. Il existe beaucoup de dipôles (ampoules, moteurs, piles, interrupteurs...). Un générateur est un **dipôle actif**, une ampoule un **dipôle passif**. Dans ce schéma, on voit que le courant circule depuis le + du générateur, passe par le dipôle et rejoint le - du générateur.

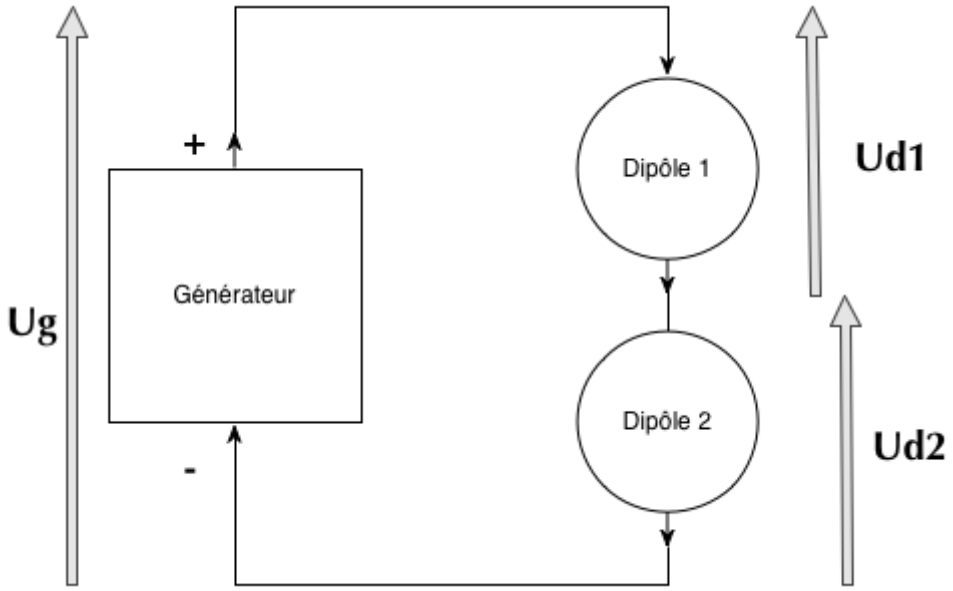


Les grosses flèches montrent les différences de potentiel. Le générateur est, pour notre rivière, la pompe qui ramène l'eau à la source ; la rivière qui coule est le reste du circuit. On imagine que la rivière passe par une cascade : le dipôle. Et bien la hauteur du système qui amène l'eau de l'embouchure à la source est la même que celle de la source à l'embouchure. Ça paraît évident. Et bien c'est pareil électriquement !

La différence de potentiel Ug est égale à la différence de potentiel Uc on a donc :

$$Ug=UcUg=Uc$$

Voici un second schéma :



Si le dipôle 1 représente une partie de la rivière et le dipôle 2 une autre partie, et bien la hauteur totale de descente est bien la somme des hauteurs des 2 parties. Ce qui donne en électricité :

$$Ug=Ud1+Ud2$$

C’est une des lois de Kirchhoff que l’on appelle **la loi des mailles.** Elle est applicable en circuit fermé et courant continu, ce qui sera notre cas.

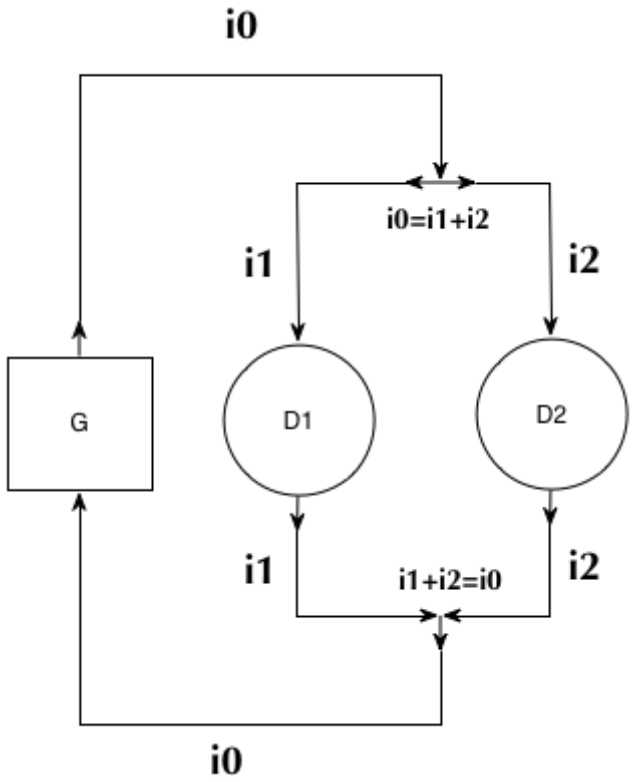
L'intensité

Bon, nous venons de voir la tension, voyons maintenant l'intensité...

L'**intensité** est la quantité de courant qui passe dans un endroit donné du circuit. Elle se mesure en **Ampères (**symbole **A)**.

C'est, pour simplifier, le nombre de particules qui passe par seconde. Pour reprendre notre rivière, ce serait le débit de la rivière à un endroit donné.

Lorsqu’un circuit électrique se sépare en deux parties, la quantité de courant se répartit dans chaque partie de ce circuit (comme la rivière lorsqu'elle est coupée par une île). On comprend bien pour la rivière, que la quantité d'eau se sépare. Et bien c'est pareil en électricité. De même lorsque les deux parties de la rivière se rejoignent, on retrouve la même quantité. C'est une autre loi de Kirchhoff : la **loi des noeuds**.



Précision : i0 est l'intensité à la sortie du générateur, i1 l'intensité dans la branche du dipôle D1 et i2 l'intensité dans la branche du dipôle D2.

Au premier croisement on a bien i0=i1+i2 et au second i1+i2=i0.

A-t-on vraiment besoin de tout ceci pour s'amuser avec l'Arduino ?

Mais c’est important que l’on fasse ce point électrique ensemble pour que vous sachiez au moins que ça existe et que brancher quelque chose sur l’Arduino nécessite quelques précautions.

Résistance et loi d'ohm

Il nous reste un dernier concept électrique à voir et la loi qui s'y rapporte.

Résistance

Les **résistances** sont très utilisées en montage électronique, mais LA résistance est plus large que ça.

La résistance, pour reprendre l'analogie avec la rivière, c'est la taille de la rivière à un endroit. Ça va donc influencer le débit. En électricité c'est un peu la même chose. La résistance est la capacité d'un matériau à s'opposer au passage du courant. Et il existe des dipôles conçus pour cela : les résistances.

Cette résistance au passage du courant se mesure en **Ohms** (symbole **Ω**).

La loi d'Ohm

Et bien il faut savoir qu'en électricité, il y a une loi qui relie intensité, résistance et tension :

Tension = Résistance x Intensité, ou en plus court :

U = R x I

avec U en Volts (V), R en Ohms (Ω) et I en Ampères (A).

Donc, si rien n'est nul, R = U/I et I = U/R.

C'est particulièrement cette loi qui va nous permettre de faire nos calculs futurs.

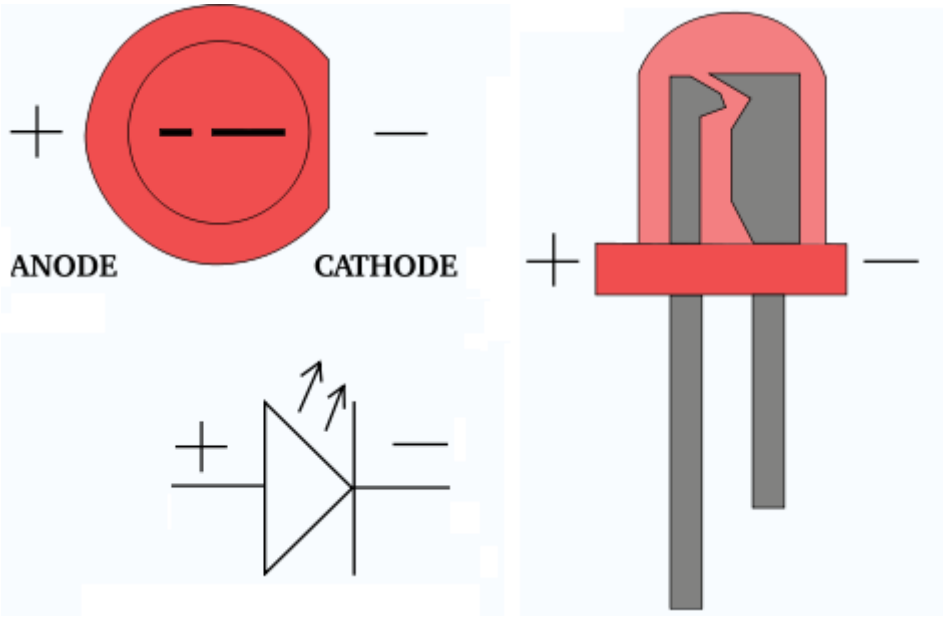
Nous allons donc attaquer la présentation de la LED ! (Non Lukas, toujours pas votre petite amie !)

Les LED

Alors les **LED** c'est quoi ? Tout d'abord LED c'est de l'anglais (*Light Emitting Diode*), en français on dit plutôt DEL (*Diode à Émission de Lumière*). Donc une LED est une diode qui produit de la lumière.

Alors une **diode** c'est quoi ?? Une diode est un dipôle (c'est-à-dire un composant électrique qui se branche avec 2 bornes) qui ne laisse passer le courant que dans **un seul sens**. On appelle ça un semi-conducteur. La diode et la LED sont des semi-conducteurs, ainsi que le transistor que nous aborderons plus loin dans le cours (et qui est la base de presque toutes les puces électroniques actuelles).

Alors je vous invite à en prendre une entre vos doigts habiles et propres, et à l'observer !



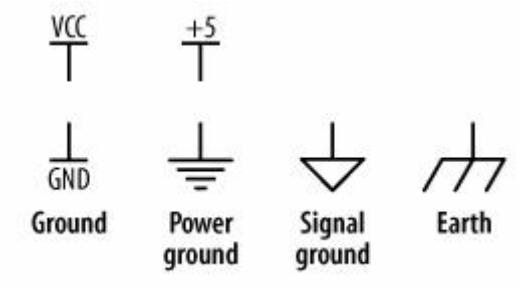
Représentations d'une LED

Vous remarquez une patte plus longue que l'autre, et un bord un peu coupé sur la base. Ces deux choses sont des repères pour pouvoir distinguer la patte qui se branche sur le + et celle qui se branche sur le - .

Si l'on veut qu'une LED s'allume, on connecte la patte la plus longue (anode ou +) du côté du 5 V et la patte la plus courte, qui part du côté coupé de la bague (cathode ou -) du côté du 0 V ou *ground*.

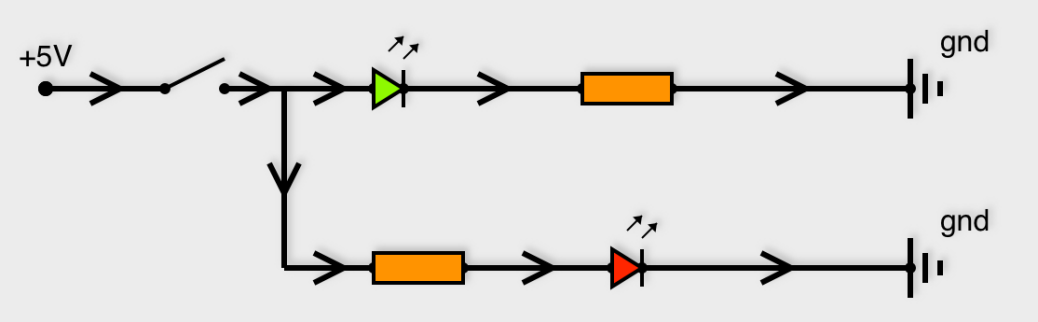
Hein ? quoi ? de quoi 0 V ou ground ???
En effet ça mérite une explication. L'Arduino n'est pas seulement un générateur de courant électrique, même s'il peut distribuer des courants de tensions données à certaines de ses bornes. Par convention, le + est la borne qui distribue un courant de 5 V de tension (ou 3.3 V dans certains cas) et le - est appelé le *ground*, c'est à dire la terre (un peu comme la prise de terre dans la prise électrique). Le + fournit une tension et le ground l'absorbe. Tout ce qui va au - est relié au ground.

Voici un exemple de symboles pour représenter le ground et le +5 V :



Symboles d'entrées électriques et de ground

Et un schéma comme vous en rencontrerez bien d'autres !



Exemple de circuit électrique avec une entrée +5V et des connexions au ground

Vous aurez compris que "gnd" veut dire ground.

Alors, sur votre Arduino, si vous l'observez, vous devriez voir

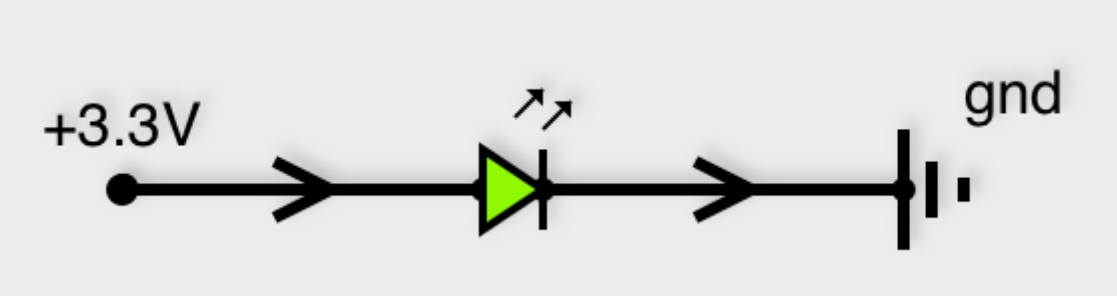
- 3 pins Gnd
- 1 pin 3.3 V
- 1 pin 5 V

Repérez-les bien car ils vont nous servir à chaque fois (au moins le 5 V et les Gnd)

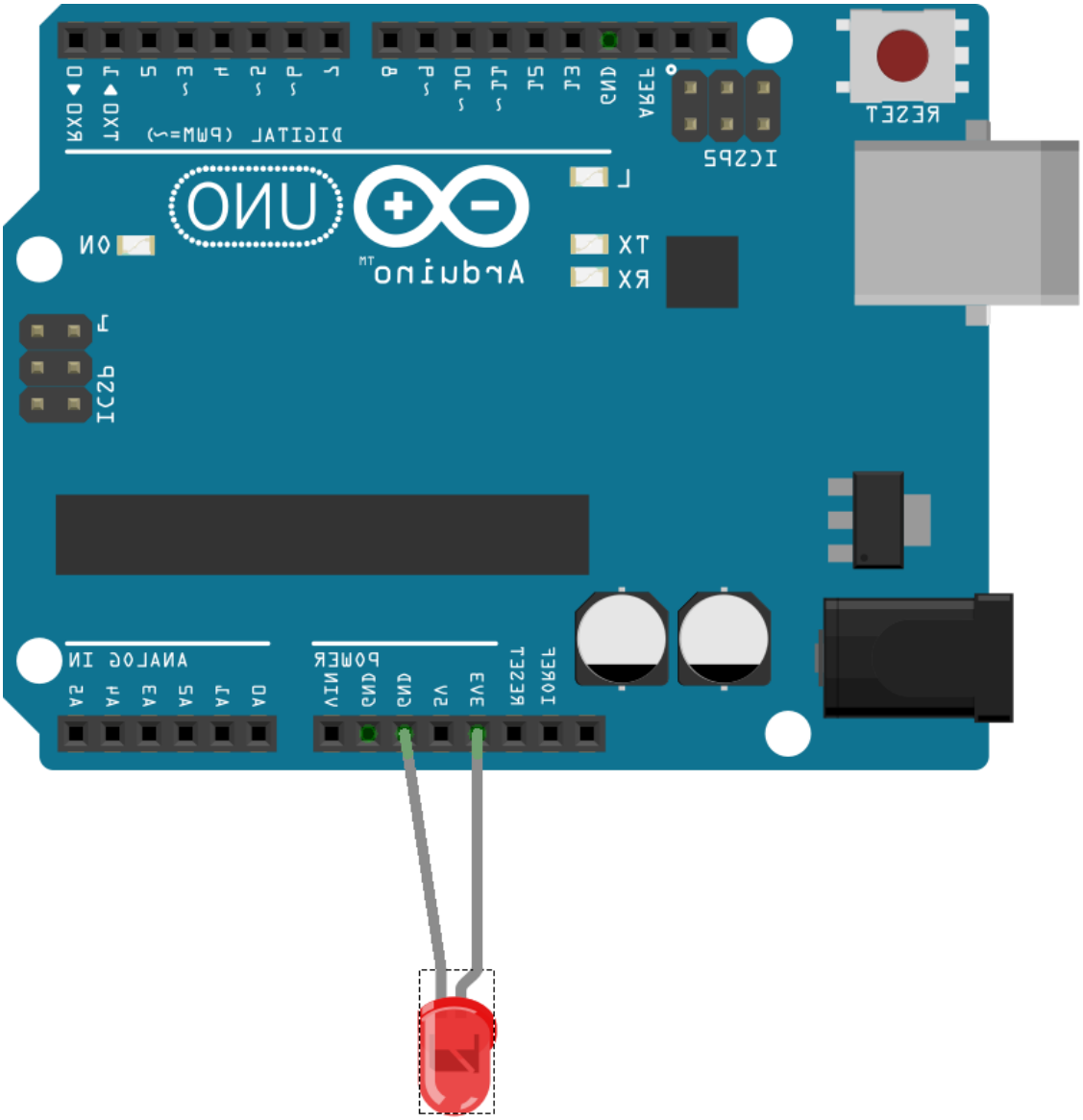
Il y a 3 bornes gnd, mais en fait elles sont reliées ensemble. Le gnd, ou ground ou terre ou masse, représente le niveau 0 V.
Alors, vous vous dites, si je veux brancher ma LED, je mets la grande patte sur le +5V, la petite patte sur le Gnd à côté et ça devrait marcher !

Oui, mais si vous faites cette expérience, branchez plutôt la grande patte sur le +3.3V, sinon vous allez griller votre diode assez rapidement. Je vous laisse faire votre expérience.

Voici le schéma électrique de ce branchement :



À partir de maintenant, je vous fournirai les schémas de l'ensemble du montage avec Tinkercad, c'est tellement plus simple !



Cool ! Elle s'allume ! Et comment on la fait clignoter maintenant ?

Pour répondre à cette question nous devons apprendre deux points importants : l'utilisation de la breadboard et la bonne manière de connecter une LED sur le +5V ou le 3.3V.

La breadboard

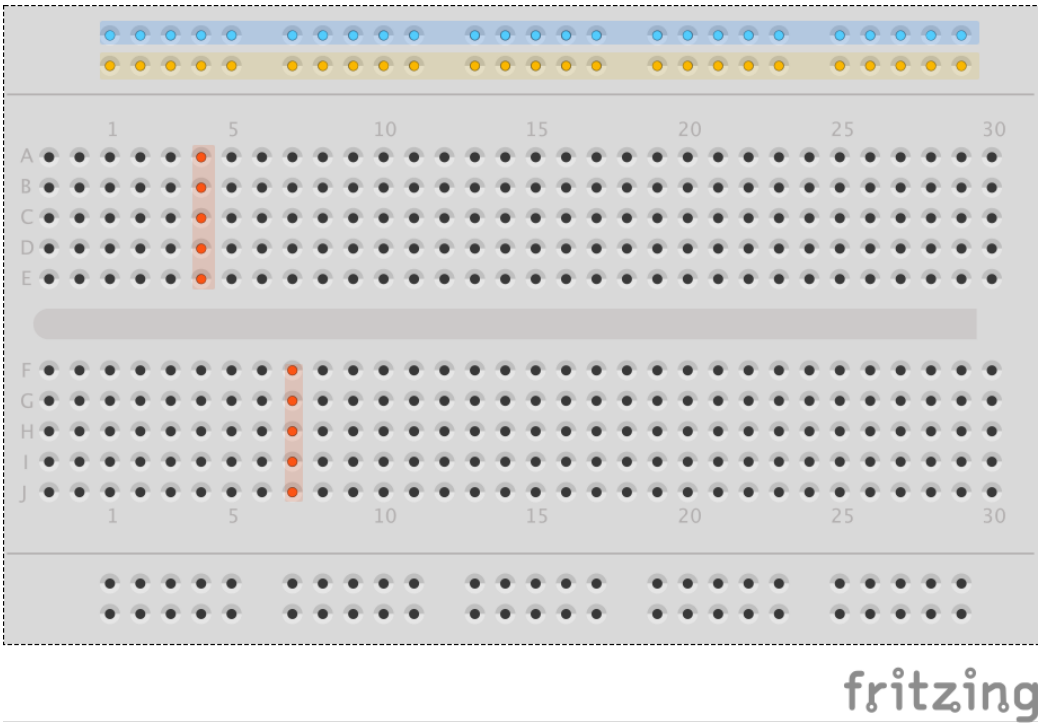
Il est temps de comprendre l'utilisation de la fameuse breadBoard !

"Des ptits trous, des ptits trous, toujours des ptits trous !"(Gainsbourg)

C'est clair qu'il y en a des trous sur ce truc, c'est pas une planche à pain, c'est une passoire !

Et bien chaque trou permet d'enficher (pour les plus vieux) ou de plugger (pour les plus jeunes) ou d'enfoncer (pour les plus bourins) les composants. Sauf que ces trous ne sont pas

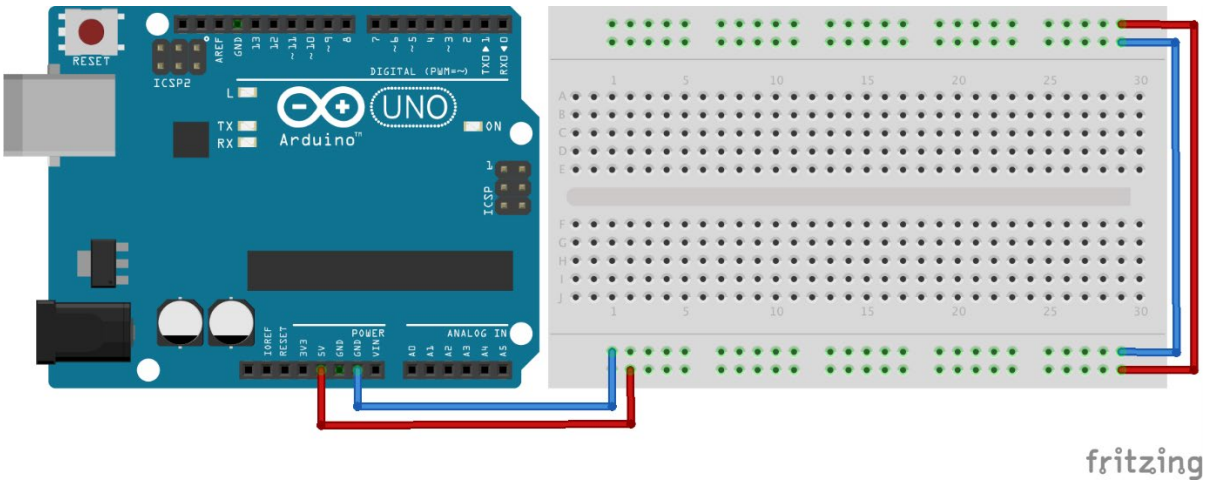
placés n'importe comment, et certains trous sont reliés à d'autres. Les trous que j'ai colorés de la même couleur sont reliés entre eux :



Liaisons entre les trous de la breadboard

Il y a donc les trous en haut et en bas, chaque ligne (ici bleue et jaune) montre des trous liés entre eux. Puis au centre, les 5 trous de **chaque colonne** (en rouge) sont liés. "Liés", ça signifie que si on connecte un composant dans un trou et un autre dans un trou lié, ils sont connectés entre eux.

Les deux grandes lignes servent souvent à relier les Vin (+5 V) et les grounds (Gnd) entre eux. Par exemple :

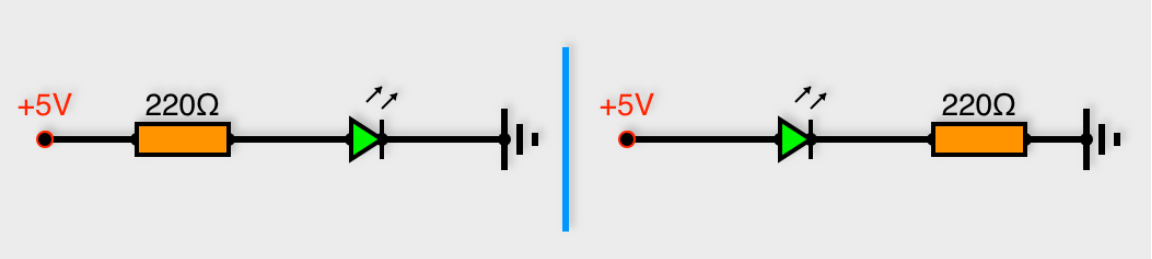


Branchement de la breadboard pour lier +5V et Gnd

Du coup tout ce qui sera connecté aux trous liés à +5 V recevra du +5 V et tout ce qui sera connecté aux trous liés à Gnd ira vers le 0 V.

Bon, la suite permet de calculer la résistance optimale pour connecter une LED au circuit. Si vous n'êtes pas fans de trop de calculs électriques, vous pouvez passer directement à la section suivante, mais retenez-que pour connecter une LED **il faut lui joindre une résistance d'au moins 100 Ω (**220Ω est souvent la valeur conseillée**)**. Plus la résistance sera grande, moins la LED éclairera.

Voici le schéma du circuit :

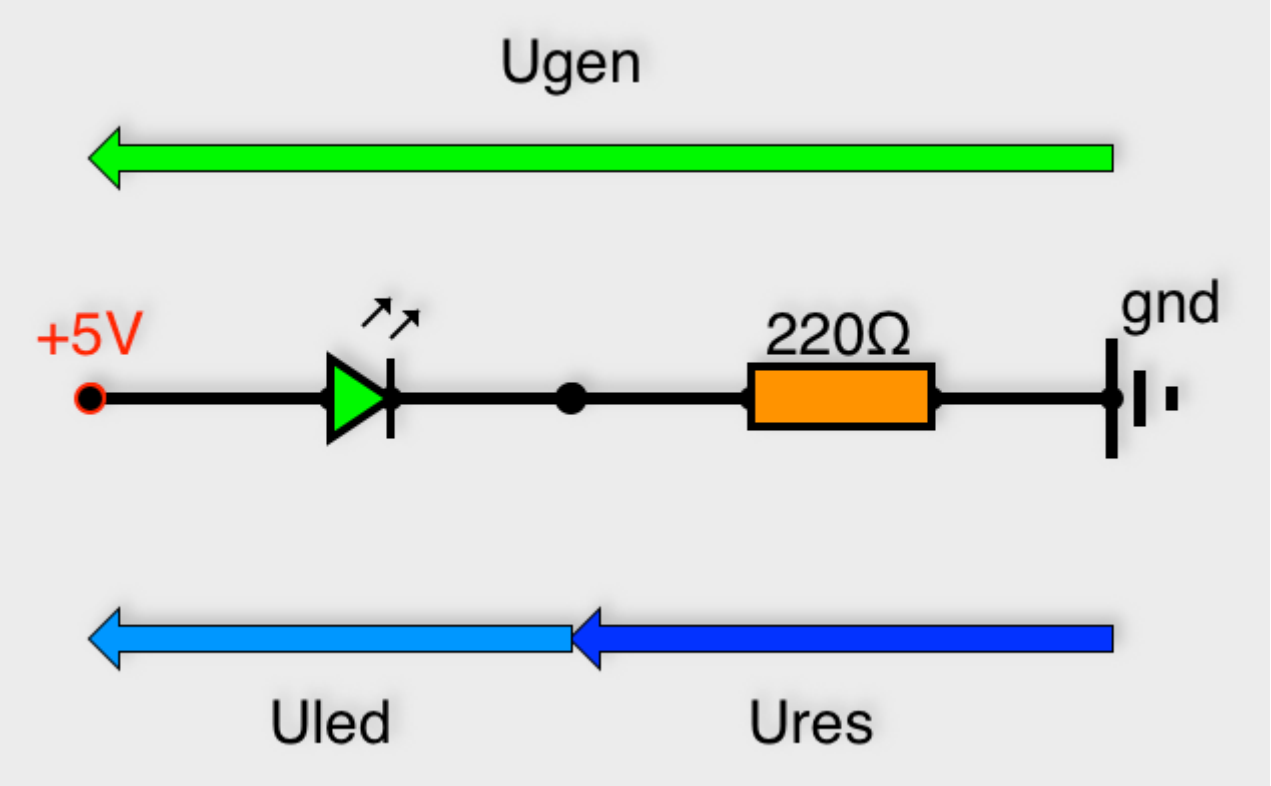


Branchement d'une LED en série avec une résistance (les deux montages sont corrects)

Calcul de la bonne valeur de résistance pour une LED

Tout matériel électronique est accompagné d'une fiche de spécifications techniques (aussi appelée *datasheet*). La LED n'échappe pas à la règle.

En la lisant, nous pouvons retenir qu'une LED produit une chute de tension entre 1 et 4 volts (en fait ça dépend des couleurs). Nous allons appeler la tension de notre LED "Uled" et la fixer à 1,8 V par exemple. La tension fournie par l'Arduino est de +5V, nous l'appellerons Ugen. La tension aux bornes de la résistance sera Ures. Voici le schéma correspondant :



D'après la loi des mailles, on a, pour le schéma du dessus :

$$Ugen=Uled+Ures$$Ugen=Uled+Ures

avec Ugen = 5 V (tension aux bornes de l'Arduino) et Uled = 1,8 V, donc :

$$Ures=Ugen−Uled$$Ures=Ugen−Uled

Si on remplace par les valeurs : Ures = 5 V - 1,8 V = 3,2 V. C'est donc la tension aux bornes de la résistance.

L'intensité de fonctionnement qui traverse une LED doit être aux environs de 20mA, soit 0,02A (ça aussi on le trouve dans la datasheet, et c'est valable pour beaucoup de LED)

Sur l'Arduino, la sortie +5V délivre un courant de 0,5 ampères (ou 500 mA) et les pins de 0 à 13 produisent du 0,04 ampères (40 mA) et un max de 0,2 ampères pour les 13 pins cumulés. Donc on est large pour une diode.
On applique maintenant la loi d'Ohm :

On a U=R x I donc R=U/I ce qui va nous permettre de calculer la résistance à utiliser.

Le calcul de la résistance à utiliser est donc R=3,2/0,02=160 Ω.

Il nous faut donc utiliser une résistance de 160 Ω pour notre exemple.

Alors le calcul précis n'est pas de la plus grande importance, en fait si vous utilisez une résistance qui se situe entre 100 Ω et 1 000 Ω, vous évitez les risques de griller la diode. Le site Arduino montre des montages avec des résistances de 220 Ω (rouge,rouge marron), j'utilise pour ma part des 100 Ω (marron, noir, marron), et si vous essayez 1 KΩ (marron, noir, rouge) ça fonctionne encore très bien.

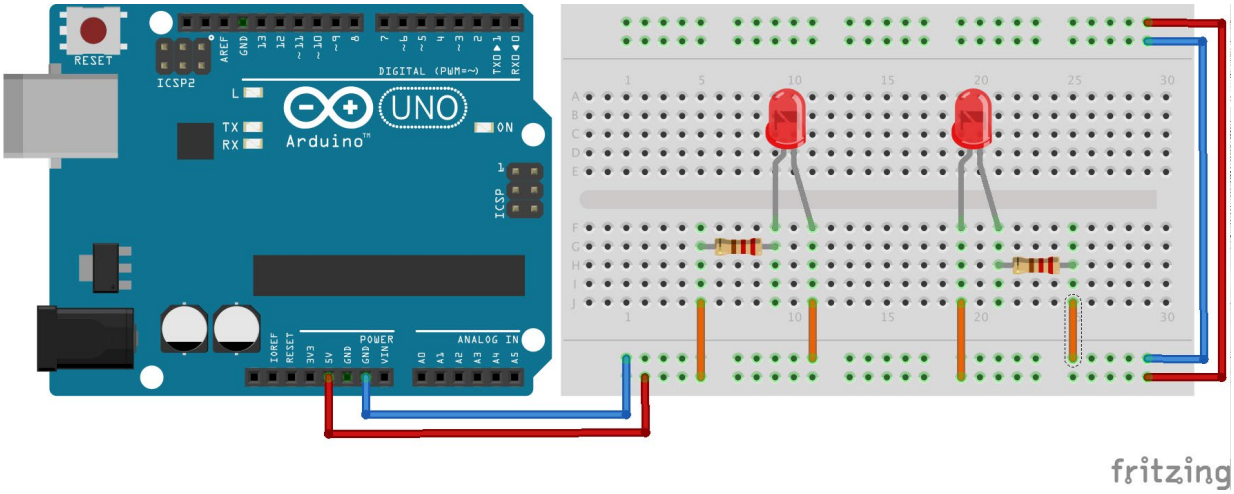
Mais pourquoi tous ces calculs compliqués alors ?

Parce que certains veulent aller plus loin, parce que certains veulent vraiment contrôler la luminosité de leur LED, et parce qu'un jour peut-être vous voudrez calculer ces données avec précisions alors vous saurez où chercher... et puis un peu de culture générale ne fait pas de mal.

Connectez une LED sur un pin (1 à 13)

Schéma de montage de LED

Voici un montage Arduino avec deux LED.



Observez ce schéma : j'ai placé la résistance avant la LED dans un cas (gauche), et après la LED dans l'autre cas (droite). Les deux montages sont corrects.
Ici, la grande patte de la LED est représentée par un petit décroché. Observez ce schéma : pour une LED, j'ai placé la résistance avant la LED, et pour l'autre après la LED. Ça n'a aucune incidence sur l'intensité qui parcourt la LED.

Mais il faut tout de même avoir en tête le parcours du courant du plus vers le moins.

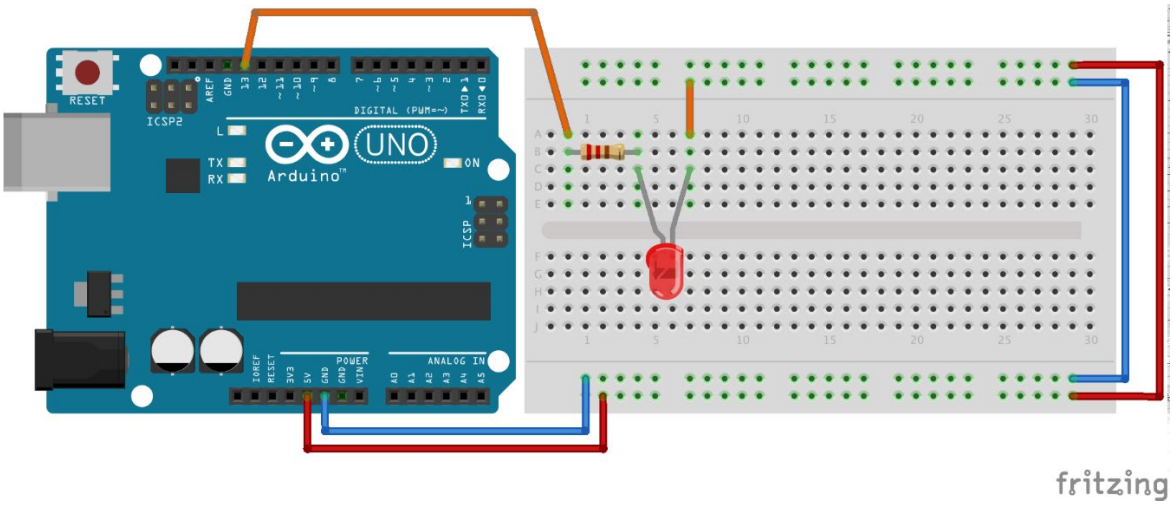
On part du +5V de l'arduino, on suit le fil rouge, qui entre dans la breadBoard, il est relié au fil orange qui monte et est connecté plus haut. Cette connexion relie le fil orange à la patte de la résistance puis sort par l'autre patte. La connexion dans la breadBoard amène le courant dans la patte + de la LED, puis traverse la LED, et sort par la patte -. Ensuite le courant passe dans la breadBoard jusqu'à l'autre fil orange, qui entre dans les trous reliés au Gnd. le courant rejoint alors le Gnd, la boucle est bouclée.

Suivre le parcours du courant est très important quand on connecte des composants. Certains sont des semi-conducteurs (comme la LED), d'autres peuvent griller s'il ne sont pas connectés correctement. Soyez donc attentifs dans vos montages.
Première méthode pour connecter une LED sur un pin

La sortie +5V fournit du courant continuellement. En revanche les pins de 1 à 13 ne fournissent du courant que par programmation. C'est tout l'intérêt de l'Arduino.

Nous allons reprendre le programme "Blink Blink" qui permettait de faire clignoter la LED 13 (celle directement sur la carte Arduino) et l'utiliser pour faire clignoter une LED branchée sur la breadBoard. La patte + de la LED est orientée vers le pin 13.

Tout d'abord, le branchement :



Si on suit le parcours du courant, il part du pin 13, il passe par la résistance, puis par la LED, puis il rejoint les trous connectés au Gnd de l'Arduino (prenez votre temps pour vérifier cela, quitte à suivre avec votre doigt le parcours).

Reprenons maintenant le programme :

```
int pinLed=13; //variable pour le numéro du pin utilisé

void setup()

{

    pinMode(pinLed,OUTPUT); //le pin 13 en mode sortie de courant

}

void loop()

{

    digitalWrite(pinLed,HIGH); // on passe le pin à +5V

    delay (1000);

    digitalWrite(pinLed,LOW); // on passe le pin à 0V

    delay(1000);

}
```

La commande`pinMode()` sert à signaler que le pin 13 va envoyer du courant (et pas en recevoir). Cette commande est incontournable.
Une fois prêt, le pin se met en +5V ou 0V grâce à la commande`digitalWrite()`.
On envoie donc du courant par programmation et plus uniquement par branchement !

En exécutant le programme, vous remarquerez que votre LED clignote ainsi que celle de la carte. La LED de la carte est liée au pin 13.
Petit exercice : Modifiez le programme et les branchements pour que la LED clignote sur le pin 7.

Vérifiez la solution !

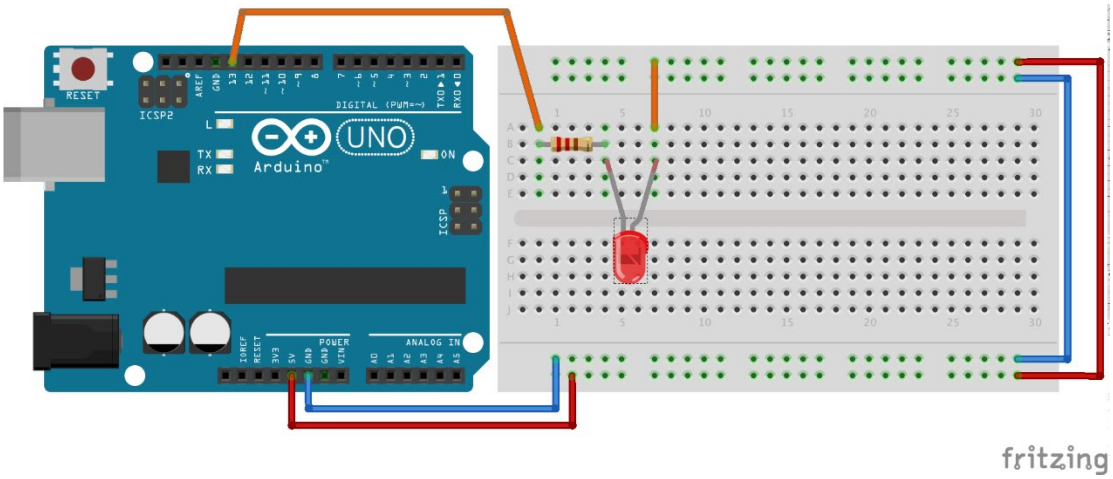
Voyons une autre façon de connecter un LED.

Une autre méthode pour ménager la carte Arduino

L'exemple précédent utilise le pin 13 (ou 7) pour fournir du courant qui s'enfuit ensuite vers le Gnd. Cette méthode de branchement est correcte, mais elle demande de l'énergie à l'Arduino. Ce n'est pas trop la spécialité de la carte... elle préfère absorber du courant qu'en fournir.

Il existe une autre façon de brancher une LED (ou autre composant mangeur de courant) qui permet de ménager notre petit Arduino.

Il faut monter le circuit à l'envers. C'est à dire que vous partez du +5V de l'Arduino, vous connectez ensuite la résistance et la LED (l'ordre importe peu) puis vous reliez le tout au pin 13. N'oubliez pas de brancher la patte + de la LED vers le +5V.



(Non c'est pas pareil ! Lukas, je fais appel à votre sens de l'observation, on a inversé le sens de la LED et la connexion sur la breadboard. Allez, concentrez-vous un peu !)

Hein ?? On envoie du +5V dans le pin 13 ? Ça va pas griller la carte ?
Et bien non, rappelez-vous que le courant ne circule que s'il y a une différence de potentiel électrique. Si on indique à l'Arduino (avec la commande`digitalWrite()` et`HIGH` d'envoyer du +5V dans le pin 13, on annule la différence de potentiel (+5V fourni des deux côtés) donc le courant ne circule pas. Puis lorsque, par programmation, on dit que le pin 13 se met à 0V (avec LOW), le pin 13 devient un Gnd et absorbe le courant (car il se met à circuler) et la diode s'allume.

Une autre différence avec le branchement précédent c'est qu'il faut adapter le programme :

Pour allumer, on doit positionner le pin sur LOW et pour éteindre sur HIGH (ça ne change rien aux instructions du programme, à part qu'elles sont données dans l'ordre inverse).

Bon je crois que le principal est dit sur les notions d'électricité, l'utilisation de la breadboard et le branchement d'une LED à l'Arduino. Passons à l'action dans le chapitre suivant avec la réalisation de deux programmes avec plusieurs LEDs...

6. Tableaux et jeux de lumière avec plusieurs LED

Vous avez maintenant un niveau en programmation et en électricité qui va vous permettre de vous amuser un peu avec les LED. Ce chapitre a pour but de vous apprendre à gagner en autonomie dans l'utilisation de l'Arduino. Il aborde aussi une sorte de variable importante : le tableau.

Vous allez monter en maîtrise en créant deux programmes ainsi que leurs montages correspondants :

- "Blink à trois" : Blink à trois, un programme qui fait clignoter 3 LED à des temps précis.
- "GuirLED" : une petite guirlande de 5 LED pour réaliser des figures lumineuses.

J'en profiterai pour aborder le stockage de variables sous forme de tableaux. On est souvent amenés à l'utiliser, mais ce n'est pas toujours simple à réaliser.

L'idéal serait que ces programmes, vous les réalisiez seul(e). Je vais donc vous aider pour le premier, mais pour le second... enfin, nous verrons.

Projet 1 : "Blink à trois"

Ce programme et le montage associé doivent permettre de faire clignoter trois LED à intervalles différents. Avec cet exercice, vous allez mettre en pratique l'ensemble des notions de programmation et de montage que nous avons vues jusqu'ici.

Voici la description du programme :

- Les trois LED sont éteintes.
- Les trois LED s'allument 1 seconde.
- Après une brève extinction de toutes les LED (1 dixième de seconde), les deux premières restent éteintes et la troisième s'allume une seconde.
- Extinction brève, puis LED n°1 et LED n°3 éteintes, LED n°2 allumée une seconde.
- Extinction brève, LED n°1 allumée et LED n°2 et LED n°3 éteintes une seconde.
- On retourne au début, mais le programme recommence avec un temps d'allumage de 0,8s (8 dixièmes), puis 6 dixièmes, puis 4, puis 2.
- Le programme recommence au début.

Avant de se lancer dans la programmation, il vaut mieux réfléchir dans un premier temps au montage et se poser les bonnes questions :

- Est-ce que je branche les LEDs avec la patte + vers le pin ou vers le +5V (nous avons vu précédemment que ça change les branchements et le code) ?
- Sur quels pins je connecte chaque LED ?

- Comment j'organise mes connexions sur la breadBoard pour que ce ne soit pas le foutoir ? (surtout pour correctement repérer le sens du courant)

Une bonne méthode est de faire les branchements sans avoir connecté l'Arduino. De vérifier si rien n'est mal connecté, puis de concevoir un petit programme de test simple pour vérifier. Ensuite il faut se poser les questions liées à la programmation :

- De quelles variables ai-je besoin, et quels types ?
- Quel nom donner à mes variables pour m'y retrouver ?
- Comment résoudre les boucles pour obtenir le résultat ?

Puis je vous conseille d'utiliser du pseudo-code, c'est-à-dire un code avec du langage humain qui permet de bien voir les boucles de programmation et de projeter les résultats.

Après tout ceci, vous pourrez vous lancer dans votre "vrai" code plus facilement. N'oubliez pas de le commenter un minimum pour qu'il soit clair et lisible.

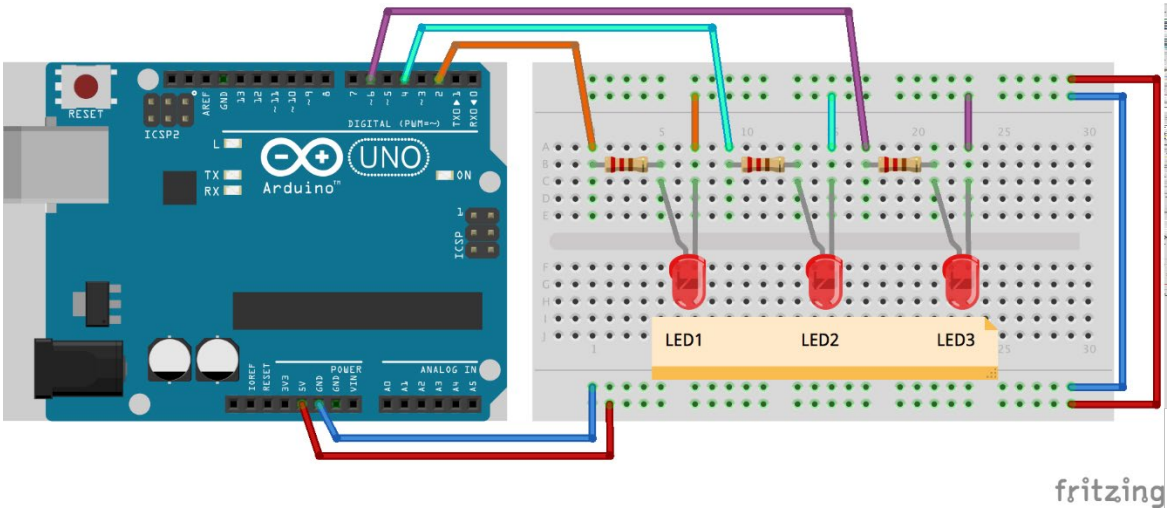
Je sais que certains impatients vont vouloir tout faire en même temps, coder directement, brancher à la va-vite. Ça n'est pas interdit, mais vous verrez qu'on gagne beaucoup de temps à réfléchir AVANT de coder...
Bon, ceux qui se sentent d'attaque pour le réaliser, et bien allez-y ! Je ne vous retiens pas ! On se retrouve à la machine à café !

Pour les autres, on va y aller plus tranquille...

Le montage et le test

Le plus simple pour le début est de connecter les pattes + des LEDs vers les pins. L'Arduino supportera la charge de connexion sans souci.

Essayez un peu avant de regarder le résultat qui suit, il n'y a aucun risque si l'Arduino n'est pas connecté.



J'ai utilisé des jumpers de couleurs différentes, ça n'influe pas sur le circuit bien-sûr, mais c'est plus clair pour le repérage.

J'ai utilisé les pins 2, 4 et 6. Vous pouvez utiliser ceux que vous préférez, sauf si vous voulez ajouter des affichages vers le moniteur, dans ce cas, n'utilisez pas les pins 0 et 1.

Vérifiez bien dans votre montage le circuit du courant, l'orientation des pattes des LEDs, les connexions au Gnd ou +5V. Ces gestes deviendront vite une routine.

Bien, maintenant, créons un programme de test pour tester (ha ?) que tout est ok. Un programme de test ne nécessite pas forcément une structure complexe, on veut juste voir si chaque LED s'allume ou s'éteint quand on envoie la commande aux pins concernés.

Là encore je vous propose d'essayer avant de regarder la solution...

```
int pinLed1, pinLed2, pinLed3; //on peut déclarer plusieurs variables

void setup()

{

  //initialisation des variables

  pinLed1 = 2;

  pinLed2 = 4;

  pinLed3 = 6;

  //initialisation des modes

  pinMode(pinLed1, OUTPUT);

  pinMode(pinLed2, OUTPUT);

  pinMode(pinLed3, OUTPUT);

  //mise à 0V de chaque pin

  digitalWrite(pinLed1, LOW);

  digitalWrite(pinLed2, LOW);

  digitalWrite(pinLed3, LOW);

}

void loop()

{
```

```
  //test allumage et repérage des lEDs

  digitalWrite(pinLed1, HIGH);

  delay(500);

  digitalWrite(pinLed2, HIGH);

  delay(500);

  digitalWrite(pinLed3, HIGH);

  delay(500);

  //on éteint tout

  digitalWrite(pinLed1, LOW);

  digitalWrite(pinLed2, LOW);

  digitalWrite(pinLed3, LOW);

  delay(500);

}
```

Les `delay()` mis dans la `loop()` permettent de vérifier le fonctionnement de chaque LED l'une après l'autre.
Si vos LED n'éclairent pas de la même façon, c'est que vos résistances sont différentes ou que les caractéristiques de chacune ne sont pas les mêmes (voir **chapitre précédent**).
Vous allez me dire : "Ouahou, c'est long non pour un programme de test ?" (Lukas, c'était façon de parler, vous n'étiez pas obligé de poser la question...).

D'une part, avec des copiés-collés bien faits, on gagne du temps, d'autre part, vous allez voir qu'on va en garder une partie pour le vrai programme...

Quelles variables, quelles boucles ?

On voit que l'objectif est finalement de faire s'allumer l'une après l'autre, les trois LED, puis de recommencer avec des`delay()` plus courts.
On a déjà les variables pour les LEDs (les nommer pinLed1, pinLed2 et pinLed3 me paraissait bien, libre à vous de donner le nom qui vous convient), il nous faut maintenant la variable qui va faire varier le temps du`delay()`.
Nous avons deux choix :

- soit c'est une variable globale (mise en début de programme) que nous allons faire varier par du code,
- soit c'est une variable de boucle et dans ce cas, c'est la boucle qui s'en charge.

Si on regarde les attentes, on demande de commencer à 1000ms (1 seconde), puis 800, 600,400, et enfin 200 et on recommence. Une boucle est donc très adaptée avec non pas un incrément (augmenter la valeur du compteur) mais un décrément (diminuer la valeur).

On peut donc imaginer une boucle du genre :

```
for (int temps=1000;temps>=200;temps=temps-200)
```

... qui veut dire : on met un compteur qui débute à 1 000 ms, qui diminue de 200 ms à chaque tour de boucle, et qui continue de faire des tours de boucles tant qu'il est supérieur ou égal à 200.

Voici le premier code que je propose :

```
// déclaration

int pinLed1, pinLed2, pinLed3; //variables des pins

void setup()

{

  //initialisation des variables

  pinLed1 = 2;

  pinLed2 = 4;

  pinLed3 = 6;

  //initialisation des modes

  pinMode(pinLed1, OUTPUT);

  pinMode(pinLed2, OUTPUT);

  pinMode(pinLed3, OUTPUT);

  //mise à 0V de chaque pin

  digitalWrite(pinLed1, LOW);

  digitalWrite(pinLed2, LOW);

  digitalWrite(pinLed3, LOW);

}
```

```
void loop()

{

    //allumage des trois LED durant 1 seconde

    digitalWrite(pinLed3,HIGH);

    digitalWrite(pinLed2,HIGH);

    digitalWrite(pinLed1,HIGH);

    delay(1000);

    //on les éteint toutes brièvement

    digitalWrite(pinLed3, LOW);

    digitalWrite(pinLed2, LOW);

    digitalWrite(pinLed1, LOW);

    delay(100);

  //Boucle de la variable temps qui diminue

  for (int temps = 1000; temps >= 200; temps -= 200)

  {

    //les trois LEDs sont éteintes

    digitalWrite(pinLed1, HIGH);//allumage LED 3

    delay(temps); // pendant la valeur de temps

    digitalWrite(pinLed1,LOW); //on éteint la 3

    delay(100); // court délai, tout est éteint

    digitalWrite(pinLed2, HIGH);//allumage LED 2

    delay(temps); // pendant la valeur de temps

    digitalWrite(pinLed2,LOW); //on éteint la 2

    delay(100);
```

```
    digitalWrite(pinLed3, HIGH);//allumage de LED 1

    delay(temps); // pendant la valeur de temps

    digitalWrite(pinLed3, LOW); // on éteint 1

    delay(100);

    //la boucle reprend

  }

  //retour au début de la loop();

}
```

Le très court délai où tout est éteint n'est pas visible à l'œil nu, mais il existe : `delay(100);`
Comme pour les programmes précédents, amusez-vous à modifier les valeurs dans la boucle pour en voir les effets et les limites visuelles.

Les tableaux

Je suis sûr que vos yeux de lynx ont remarqué avec grande sagacité que le code précédent présente des répétitions : on réalise la même opération pour chaque pin. Il existe plusieurs moyens de créer du code qui se répète en l'utilisant plusieurs fois et en changeant les variables. Ici nous allons le faire à l'aide de tableaux.

Qu'est-ce qu'un tableau en programmation ? (Oui Lukas, en programmation, sinon je sais que vous savez ce que c'est !)

C’est une façon de stocker des variables dans un même endroit au lieu de les créer séparément et leur donner des noms différents.

Les tableaux ne peuvent stocker **que des variables de même type** !
Prenons l'exemple de nos variables de pin dans le code précédent :

- Elles sont toutes de type "int" ;
- Seuls leurs numéros sont différents (pinLed1, pinLed2, pinLed3)
- On les utilise par la suite pour des opérations similaires.

On peut donc facilement utiliser un tableau.

Plutôt que de donner un nom différent à chaque variable de pin, nous allons donc les regrouper dans un tableau nommé “pinLed” par exemple, et il suffira de les repérer dans ce tableau à l’aide d’un numéro.

Comment déclarer un tableau ?

Il existe plusieurs façons de déclarer un tableau, mais avant tout il vous faut repérer commente écrire des crochets : [et].

- Pour les utilisateurs de Windows, il faut appuyer sur :

`Alt Gr` + `(` ou `Alt Gr` + `)`

- Pour les utilisateurs de mac, c'est la combinaison :

`Alt` + `⇧` + `(` ou `Alt` + `⇧` + `)`

Il faut maintenant déclarer le tableau pour que l'IDE sache qu'on va utiliser un nom générique pour plusieurs variables. Et bien on le lui dit avec ces fameux crochets !

Dans notre cas c'est un tableau avec des "int". Il n'y aura que trois valeurs à stocker : la valeur du pin pour la LED 1, celle pour la LED 2 et celle pour la LED 3. À la déclaration d'un tableau on doit dire à l'IDE combien de valeurs doit stocker le tableau. Donc pour nous : 3.

Voici le code de déclaration du tableau :

```
int pinLed[3];
```

Ça veut dire : ménage un espace dans ta mémoire pour stocker pour un tableau de type "int" avec le nom générique pinLed qui contiendra 3 valeurs.

Comme l'IDE connaît la taille prise par un "int" et le nombre de valeurs à stocker, il prépare en mémoire une place pour un tableau de 3 int.
Ensuite on doit remplir le tableau. Là encore, on a plusieurs choix :

- Soit on remplit le tableau à la déclaration, on utilise alors des accolades et on sépare les valeurs par des virgules :

```
int pinLed[3]={2,4,6};
```

- Soit on remplit le tableau valeur par valeur, dans ce cas on indique les valeurs à ranger dans chaque “case” numérotée du tableau :

```
int pinLed[3]; // déclaration du tableau

pinLed[0]=2; // on donne la valeur 2 à la première valeur

pinLed[1]=4; // 4 à la deuxième

pinLed[2]=6; // 6 à la troisième
```

Vous remarquerez que nous mettons entre crochets la position de la valeur dans le tableau. Ce numéro commence toujours à 0. La première valeur d'un tableau est donc `tableau[0]` et pas `tableau[1]`. C'est une erreur fréquente quand on commence à programmer.
Et comment on va chercher ces valeurs ensuite ?
Ha, je serai toujours impressionné par la pertinence de certaines questions !

Et bien on appelle (ou utilise) une valeur d'un tableau en combinant le nom générique du tableau et la position de la valeur, par exemple pour utiliser la valeur en position 1 on fera :

```
pinMode(pinLed[1],OUTPUT);
```

Le programme ira chercher la valeur située en position 1 du tableau (4 dans notre cas) et l'utilisera comme une variable.

Je vous donne donc le code du programme “Blink à trois” modifié pour ranger les variables de pin dans un tableau... Prenez votre temps pour le comprendre.

```
/*
Pour ce programme on utilise des LEDs connectées sur les pins 2,4 et 6
*/

int  pinLed[3] = {2, 4, 6}; //déclaration et initialisation du tableau
                            //avec les valeurs des pins
void setup()
{
  //Boucle d'initialisation des modes et mise à 0V
  for (int i = 0; i < 3; i++) // i va nous servir pour parcourir le tableau
  {
    pinMode(pinLed[i], OUTPUT); //on utilise les valeurs du tableau
    digitalWrite(pinLed[i], LOW); // l'une après l'autre
  }

}
void loop()
{
  //on allume les 3 LED (ici en utilisant une boucle)
  for (int i = 0; i < 3; i++) // on parcourt le tableau
  {
    digitalWrite(pinLed[i], HIGH);// on allume
  }
  delay(1000); //pendant 1 seconde
```

```
  //puis on les éteint brièvement (ici en utilisant leur position dans le tableau)

  digitalWrite(pinLed[0],LOW); // on éteint la 1

  digitalWrite(pinLed[1], LOW); // on éteint la 2

  digitalWrite(pinLed[2],LOW); // on éteint la 3

  delay(100);//délai bref



  //Boucle pour diminuer le temps d'allumage des LED

  for (int temps = 1000; temps >= 200; temps -= 200)

  {

    //Les trois LED sont éteintes

    for (int i = 0; i < 3; i++) // t pour parcourir le tableau

    {

      digitalWrite(pinLed[i], HIGH);//on allume la LED correspondante

      delay(temps); // pendant la valeur de temps

      digitalWrite(pinLed[i], LOW); //on éteint la correspondante

      delay(100); // court délai pendant lequel les 3 LED sont éteintes

    }

    //on a allumé les 3 LED pendant une durée "temps", la boucle reprend avec une valeur
"temps" décrémentée

  }

}
```

Le programme est bien plus court. J'attire votre attention sur la boucle qui permet de "lire" un tableau. On fait varier une variable (tiens donc...) de la position 0 du tableau à sa valeur maximum. Du coup on peut aller chercher les valeurs pour chaque position. On peut aller les chercher directement avec leur numéro de position (ligne 26 à 28) qui est dans le code pour l'exemple.

Attention, la position de la dernière case d'un tableau n'est pas la taille du tableau, mais sa taille moins 1. C'est parce que la première case d'un tableau est numérotée à 0, pas à 1. Si

vous comptez vos doigts en commençant par 0 vous trouverez 9 et non 10. C'est pour cela que dans la boucle on utilise le signe< et non<= .
Nous aurons bien des fois l'occasion d'utiliser des tableaux, dès le programme suivant d'ailleurs, donc ne vous alarmez pas, ça va finir par rentrer.

Le point machine à café

Bon, pour ceux qui ont essayé seul(e)s et qui ont réussi, je vous dis bravo ! C'est important d'aboutir par ses propres moyens (vous Lukas, c'est rarement propre et souvent moyen, mais ne vous découragez pas).

Jetez un œil tout de même à la solution de code que j'ai proposée, car les tableaux solutionnent souvent bien des choses. Attention, il ne s'agit pas d'utiliser un marteau pour écraser une mouche !

Je m'explique : un programme est un bon programme quand il répond à la demande, dans un temps court avec un nombre de lignes de code suffisant. Chercher le moins de ligne de code possible est un travail intéressant, mais cela n'aboutit pas toujours au résultat le plus performant.

Chaque test et opération qu'effectue la machine prend un certain nombre de cycles d'horloge (souvenez-vous que votre Arduino est cadencé à 16MHz). Certains tests sont plus rapides que d'autres.

Il est tout à fait possible de tester le temps pris par une portion de code, nous verrons cela quand nous utiliserons les fonctions liées au temps et des programmes plus complexes (dans le cours de perfectionnement). Je vous ai tout de même fourni un exemple dans un chapitre précédent.

Et maintenant, c'est parti pour le deuxième programme de ce chapitre, la GuirLED !

Projet 2 : Une GuirLED

Le programme que je vous propose de réaliser utilise 5 LED. Je vous laisse choisir les couleurs que vous préférez. L'objectif est de réussir à créer des illuminations variées un peu comme une guirlande.

On essaiera dans les montages de positionner les 5 LED en une ligne assez rapprochée (contrainte de montage). Puis nos LED devront effectuer des "figures" lumineuses variées. Je vais vous proposer 4 figures pour le moment, vous inventerez celles qui vous conviennent !

- Les LED s'allument de droite à gauche une par une ;
- Les LED s'allument de gauche à droite une par une ;
- Les LED s'allument toutes puis s'éteignent de droite à gauche ;
- Les LEDs s'allument toutes puis s'éteignent de gauche à droite.

Je vous conseille vivement d'utiliser un tableau pour stocker les valeurs des pins de chaque LED. Utiliser un tableau de variables booléennes pour les états d'allumage vous sera également très utile !

Un tableau peut lui-même contenir des tableaux, à condition qu'ils contiennent tous le même type de variables. On obtient alors un tableau à double entrée.

On déclare un tableau de tableau en faisant :

```
int tableau[5][5]= {2,4,5,7,9,
                    3,4,6,8,0,
                    3,4,1,2,5,
                    4,5,6,8,3,
                    2,2,2,4,5};
// on doit déclarer le tableau en donnant les valeurs toutes à la suite.
```

Le fait de disposer chaque sous-tableau dans une ligne est surtout visuel et nous permet de nous y retrouver. Pour l'IDE, la déclaration est une succession de valeurs.

`tableau[3][4]` signifie : aller chercher la valeur à la position 4 dans le sous-tableau qui est en position 3. Ou pour le dire autrement, la valeur de la 4ème ligne (car les lignes commencent à 0), et 5ème colonne (car les colonnes commencent à 0).

Il faut donc deux boucles pour parcourir un double tableau. La boucle qui parcourt les positions des premiers crochets, et celle qui parcourt les positions des seconds crochets.

Imaginons maintenant un tableau qui contient les états de chaque LED en fonction de ce qu'on veut faire, prenons l'exemple de l'allumage de droite à gauche. Le 1 signifie allumé, le 0 éteint :

Séquence	Led1	Led2	Led3	Led4	Led5
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	1	0	0
4	0	1	0	0	0
5	1	0	0	0	0

Ce tableau ci-dessus peut être représenté dans notre code par un tableau à double entrée : un tableau de séquences, qui contient un tableau de booléens pour chaque séquence. C’est donc

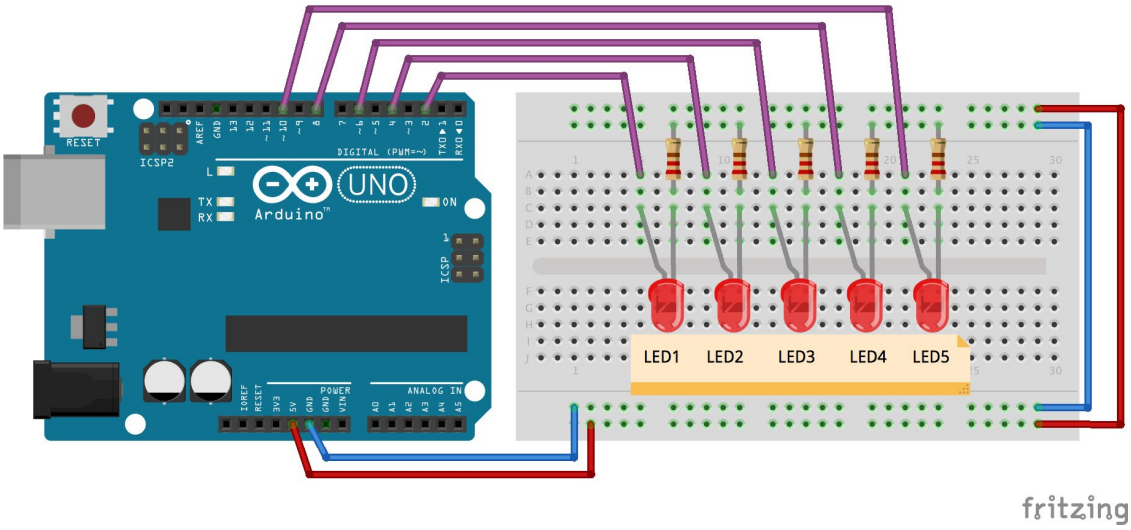
un tableau qui contient 6 tableaux (un tableau par séquence de 0 à 5), avec chacun des 6 tableaux qui contient 5 booléens (un booléen par LED).

On peut donc utiliser ce tableau à double-entrée pour mettre à jour chaque pin en position HIGH ou LOW.

Vous pouvez mettre 1 à la place de HIGH et 0 à la place de LOW dans la fonction digitalWrite(). En effet, l'état LOW est un état bas qui correspond à un 0 (dit 0 logique) et HIGH à un état haut qui correspond à un 1 (dit 1 logique).
Réaliser ce programme seul demande un gros effort de compréhension de boucles et de tableaux. Mais si vous êtes motivés, c'est tout à fait faisable. Bien sûr, vous allez devoir faire travailler votre matière grise pour y arriver. Mais je suis sûr que vous brûlez d'envie d'essayer pour devenir de plus en plus autonomes sur vos futurs projets !

Nous verrons plus tard comment utiliser des fonctions, qui vont encore nous faciliter la vie, mais ce n'est pas l'objet de cet exercice.
Solution

Voici le montage proposé pour la GuirLED, vous pouvez modifier les écarts entre les LED si ça vous chante et utiliser la résistance pour vous connecter directement au Gnd depuis la LED.



Voici maintenant le programme correspondant.

```
/*

On utilise pour ce programme 5 LED

connectées sur les pins 2,4,6,8 et 10

*/
```

```
int pinLed[5]={2,4,6,8,10}; // Tableau listant les pins

//Tableau à double entrée listant l'état (booléen 1 allumé, 0 éteint) des LED à chaque
séquence

boolean affichage[25][5]={

  0,0,0,0,0,

  0,0,0,0,1,

  0,0,0,1,0,

  0,0,1,0,0,

  0,1,0,0,0,

  1,0,0,0,0,

  0,0,0,0,0,

  1,0,0,0,0,

  0,1,0,0,0,

  0,0,1,0,0,

  0,0,0,1,0,

  0,0,0,0,1,

  0,0,0,0,0,

  1,1,1,1,1,

  1,1,1,1,0,

  1,1,1,0,0,

  1,1,0,0,0,

  1,0,0,0,0,

  0,0,0,0,0,

  1,1,1,1,1,

  0,1,1,1,1,
```

```
  0,0,1,1,1,

  0,0,0,1,1,

  0,0,0,0,1,

  0,0,0,0,0};



void setup() {

  for (int i=0;i<5;i++)

  {

    pinMode(pinLed[i],OUTPUT);

    digitalWrite(pinLed[i],LOW);

  }



}



void loop() {

  for (int i=0;i<25;i++) // boucle de séquence d'affichage

  {

    for (int p=0;p<5;p++) // boucle pour chaque pin

    {

      boolean etat=affichage[i][p]; // on va chercher l'état pour le pin

      digitalWrite(pinLed[p],etat); // on met le pin concerné à l'état

    }

    //tous les pins sont dans l'état de la séquence en cours

    delay(300); //petite pause d'affichage
```

```
    // on passe à la séquence suivante

  }

  // fin des séquences et on repart au début de la loop()



}
```

Vous observerez l'initialisation du tableau des séquences d'affichage, il ne faut pas hésiter dans un cas comme celui-ci à organiser correctement votre affichage pour vous y retrouver, on aurait pu écrire le tableau comme suit :

```
boolean affichage[25][5]={0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,

1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,

0,0,0,0,0,1,1,1,1,1,1,1,1,0,1,1,1,0,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,

1,1,1,1,1,0,1,1,1,1,0,0,1,1,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0};
```

Mais on voit bien qu'on ne voit rien !

Le tableau`affichage` contient bien, dans les deux cas, 25 séries de 5 booléens.
Amusez-vous à expérimenter avec les tableaux et les valeurs dans ce programme. Passez à 10 LED et faites des séquences plus complexes.

Rien ne vous empêche de positionner les diodes différemment (en carré, cercle, ou autre).

Vous voyez que déjà, avec peu de matériel, on peut correctement se martyriser le cerveau, pour finalement un résultat assez sympa !

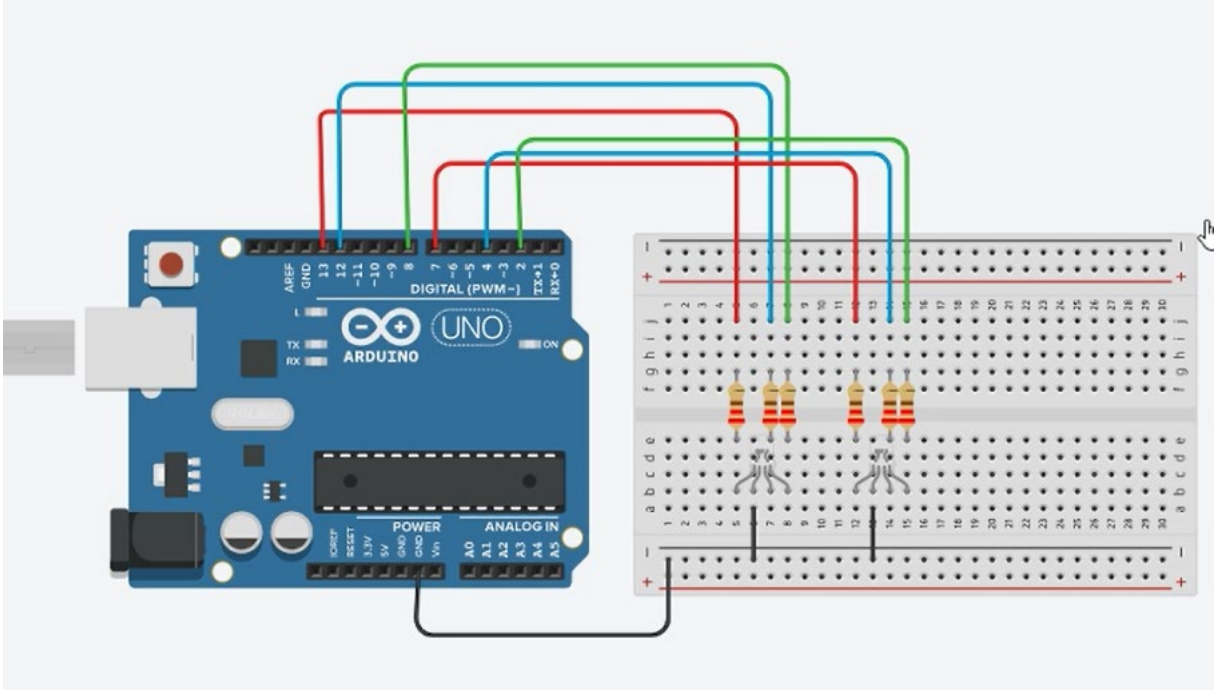
Vous voici à la fin de la première partie de ce cours. Félicitations !

Dans la seconde partie, vous allez pouvoir ajouter de nouveaux composants et matériels à votre Arduino et bien sûr de apprendre de nouvelles instruction pour le programmer !
D'ailleurs, le premier chapitre va vous donner des boutons !

7. Cotation de la première partie

7.1 Devoir 1 à faire en classe avec le matériel et sur tinkercad

Utiliser deux diodes RGB et réaliser une séquence lumineuse en faisant fonctionner toutes les secondes sur une des trois couleurs fondamentales. (Red Green Bleu)



1. créer une séquence qui fait du rouge bleu vert avec un delais de 0.5 secondes entre les couleurs et cela sur les deux RGB chacune à leur tour.

2. lorsque la RGB n'est pas utilisée mettre une couleur JAUNE dessus en faisant l'association de deux couleurs.

7.2 Quizineur : Réaliser la première partie du quizineur

https://www.heh.be/quizineur/parcours.php?sujet=17

7.3 Vérifications de connaissances sur le Ecampus