

PROTOTYPAGE ARDUINO

1. Introduction.

Nous allons pendant ce très long TD travailler sur la plate-forme de prototypage électronique Arduino. Cette plate-forme permet de créer de l'appareillage électronique interactif de manière simple.

Je vous demanderai de faire preuve de mesure et de délicatesse lors des manipulations, cela reste fragile. Je vous conseille de travailler à deux, voire trois, au moins au début. Vous rangerez en fin de séance les composants que vous aurez utilisés dans les bonnes cases, sous peine d'en être privé la fois d'après !

Enfin, si ces TD vous plaisent, vous pouvez prolonger l'expérience chez vous. Un Arduino coûte 20 euros, une boîte de composants électronique (kit de débutants pour Arduino par exemple) dans les 40 euros. Et si vous n'avez pas peur de l'anglais, un très bon bouquin : Arduino Cookbook, Michael Margolis, éditions O'Reilly (il existe aussi des livres en français, ainsi que des tutoriels sur le net, plutôt moins pratiques à mon avis). Vous pouvez aussi utiliser des modules TinkerKit ou Grove, c'est un peu plus cher et un peu plus simple.

Deux catégories de composants sont à distinguer: capteurs et actionneurs. Un capteur réagit en fonction d'un phénomène physique (température, luminosité, humidité, champ magnétique etc.). Un actionneur produit un phénomène physique ; on utilisera principalement des diodes ici. Certains composants ont les deux fonctionnalités : certains micros sont aussi des haut-parleurs.

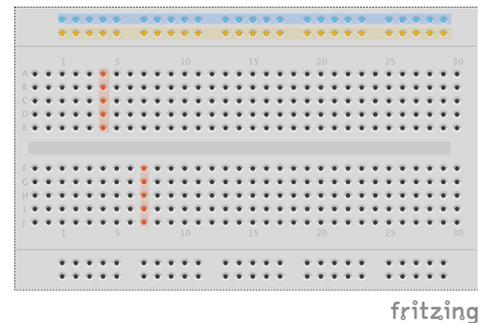
2. Première approche.

a. Mono-guirlande de Noël.

Matériel : lisez d'abord la description des composants ci-dessous. Une breadbord, une DEL, une résistance de 220 Ohms, deux câbles male/male (un rouge et un bleu si possible, surtout pour le rouge).

La *breadbord* est une planche de montage expérimentale (ou de prototypage). Elle se présente sous la forme suivante, où les trous de la même couleur sont reliés entre eux

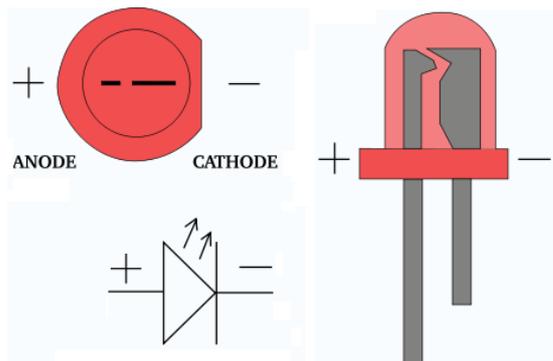
Les trous d'en haut/bas servent souvent pour le courant. Par convention, on met en général un fil rouge/marron pour l'entrée, toujours un fil bleu pour la sortie et en général un fil bicolore pour la terre pour le 220 V. La couleur rouge/marron est impérativement respectée car dangereuse, notamment avec le 220 V : ne touchez **jamais** un fil rouge ou marron dans une prise ou un interrupteur sans avoir d'abord coupé le courant !



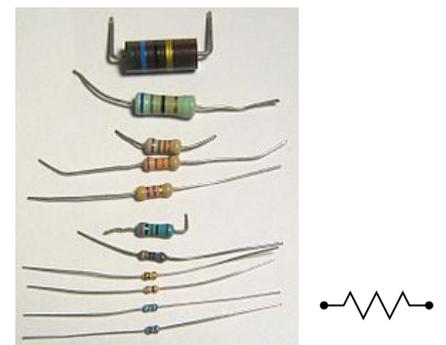
La diode est un composant électronique qui ne laisse passer le courant que dans un sens. La diode électroluminescente émet en plus de la lumière. Il faut donc la brancher correctement : repérer la patte plus courte et le côté coupé du côté négatif (cathode).



Des DEL



Vue d'une DEL avec le sens, et symbole



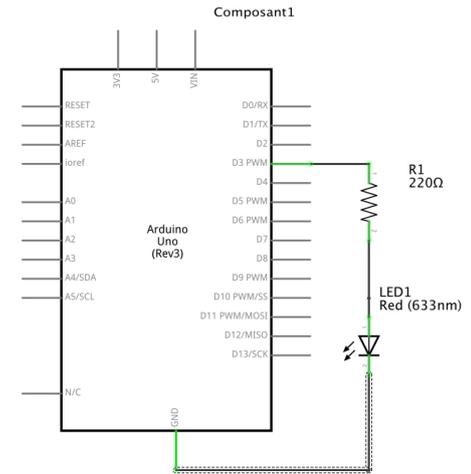
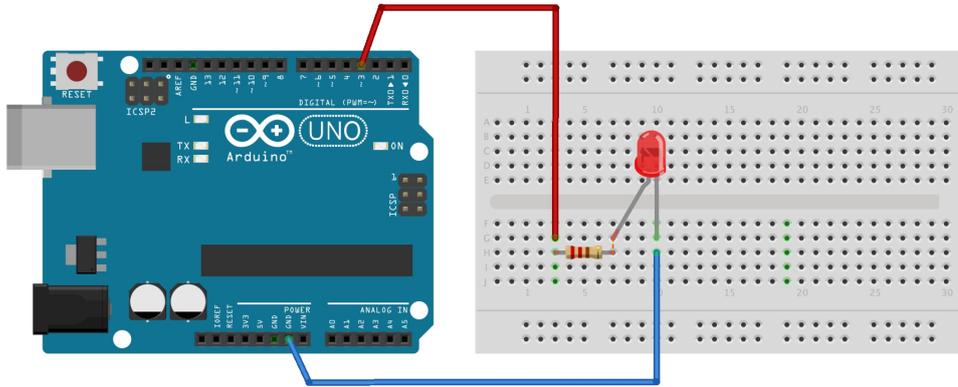
Diverses résistances, symbole

Il est nécessaire de mettre une résistance avant ou après la diode, sinon le courant est trop fort et va la griller à terme. Un calcul permet de trouver une valeur de 160 ohms. Dans les faits, une résistance entre 100 et 1k ohms est correcte. On met souvent 220 ohms pour une led. Plus la résistance est forte moins la diode émettra de lumière.

Pour la lecture de la valeur d'une résistance, voir annexe 1

Nous allons utiliser les broches (pin en anglais) 0 à 13 pour brancher les composants. En effet, on peut programmer les broches 0 à 13 pour envoyer le courant quand on le souhaite. Dans les faits, on évite d'utiliser les connexions 0 et 1, qui peuvent servir au port série (que l'on verra ultérieurement) Pour simplifier, on branchera directement le courant sur la colonne de trous où sont les composants, sans passer par les trous du bas (ou du haut).

D'où le montage que l'on réalise (vous pouvez prendre n'importe quelle connexion, sauf 0 et 1) :



fritzing

fritzing

Remarque : on peut aussi monter la diode à l'envers : dans le programme suivant, elle sera alors éteinte en mode HIGH et allumée en mode LOW. D'après OpenClassrooms, ce montage économise la carte.

b. *Le code.*

Lancez le programme Arduino sur l'ordinateur. Ce programme est un IDE : environnement de développement. C'est dans ce logiciel qu'on tape les programmes, et qu'on les envoie à la carte Arduino.

Vous devriez pouvoir choisir la langue dans les préférences. Dans outils>carte (Tools>board en anglais), cochez Arduino Uno/Génuino. Allez dans Outils>Port série, regardez quels sont les ports présents. Branchez ensuite la carte arduino via un câble USB. Puis retournez dans Outils>Port Série, vérifiez si la carte apparaît et cochez le port en question. Si elle n'apparaît pas, branchez la carte sur un autre port et recommencez.

Captures d'écran et quelques explications supplémentaires :

<http://arduino.cc/fr/Main/DebuterInstallationWindows>.

Tapez le programme figurant dans la première colonne, dans l'IDE Arduino, en respectant absolument les majuscules et symboles présents (il n'y a pas de passage à la ligne dans le commentaire) :

Programme	Explications
<code>// Programme écrit par Génial Genius</code>	Cette ligne est un commentaire, elle commence par //. On massacre allègrement l'orthographe en ne mettant aucun accent, aucune cédille etc. dans les programmes informatiques. Pour des commentaires de plusieurs lignes, on les écrit entre /* et */
<code>void setup() {</code>	Le « setup » est l'initialisation du programme, qui ne sera <u>exécuté</u> qu'une <u>seule fois</u> au démarrage. On ouvre le setup avec une accolade
<code> pinMode(13, OUTPUT);</code>	Si on a branché la diode sur le pin 13, alors on déclare la connexion 13 comme étant une sortie de courant.
<code>}</code>	Chaque ligne de programme se finit par un point-virgule.
<code>void loop() {</code>	On décale les instructions, d'une tabulation ou de quatre espaces, à chaque fois que l'on ouvre des nouvelles accolades.
<code> digitalWrite(13, HIGH);</code>	Fermeture du setup
<code> delay(1000);</code>	Boucle principale, <u>exécutée à l'infini</u>
<code> digitalWrite(13, LOW);</code>	On envoie du courant dans la broche 13.
<code> delay(1000);</code>	On attend 1 seconde (1000 millisecondes)
<code>}</code>	On coupe le courant dans la broche 13.
	On attend 1 seconde
	On ferme la boucle principale

Ce programme est écrit dans une variante du langage de programmation C. Le C est un langage très rapide, mais qui n'est pas le plus simple à écrire.

Sauvegardez votre programme, puis téléversez-le dans la carte avec le bouton . Et là c'est MAGIQUE.

Ou alors ça plante : vous avez des commentaires en orange dans la partie en bas de la fenêtre de l'IDE. Si c'est un

problème « not in sync », c'est que la carte Arduino n'est pas reconnue sur le port série. Reprenez le début de ce paragraphe. Sinon, c'est probablement un problème de syntaxe (majuscule, point virgule oublié, etc.)

Remarques :

- Les fonctions/méthodes apparaissent dans ce programme sous la forme `void nom_de_fonction ()`. `void` équivaut à `None`; si par exemple la fonction doit renvoyer un entier alors il y aura `int` à la place de `void`.
- La fonction `setup` est l'initialisation du programme, elle contient les instructions qui ne sont réalisées qu'une seule fois.
- La fonction `loop` contient les instructions qui sont exécutées en permanence, c'est-à-dire les instructions nécessaires à la fonctionnalité de l'outil que vous avez créé.
- Les fonctions sont suivies d'instructions contenues entre deux accolades `{}`. Les accolades correspondent à l'indentation en python, qui n'est donc pas nécessaire ici (mais en fait c'est indispensable pour des raisons de lisibilité !). Tous les groupes d'instructions devront être entre accolades : après un `if`, un `while`, etc. Toutes les instructions se terminent par un point-virgule.
- Autre différence importante par rapport à Python, les variables doivent être déclarées en début de programme ou de fonction, avec leur type. Il en est ainsi de :
 - `int n` déclare un entier `n` signé (2 octets... ou 4 ?)
 - `float x` déclare un flottant (4 octets)
 - `boolean b` déclare un booléen (Vrai ou Faux, en théorie 1 bit, en pratique 1 octet)
 - `char c` déclare un caractère ou un entier signé entre -128 et 127 (1 octet donc).
 - `byte b` est un octet non signé
 - `string s` représente une chaîne de caractères. C'est un tableau en fait.
 - `void` est utilisé quand une fonction ne renvoie rien
- le tant que s'écrit `while (compteur > 0) { instructions }`
- idem pour le « si » et le « pour » :
 - `if (booléen == True && autreTest != 0) { instructions } else {}`
 - `for (int i = 0 ; i < 4 ; i++) { instructions }` correspond au `for i in range(4)`
 - le « et » est `&&`, le « ou » est `||`
- Vous remarquerez qu'il n'y a pas de point virgule en fin de ligne quand il y a des accolades, qui ouvrent alors un bloc d'instructions.
- Certaines instructions sont spécifiques à Arduino, il en est ainsi :
 - du `delay(1000)` qui dit au processeur de ne rien faire pendant mille millisecondes.
 - Du `digitalWrite(13,HIGH)` qui allume la led en envoyant du courant. Vous devinerez peut-être tout seul ce que fait `digitalWrite(13,LOW)` 😊
- Enfin, autre différence fondamentale avec Python, le C est un langage compilé. La compilation est signalée lorsque l'on appuie sur le bouton de téléversement, elle peut se faire indépendamment, ne serait-ce que pour vérifier la syntaxe. Je vous rappelle que la compilation traduit tout le code en langage machine avant exécution du programme, alors que l'interprétation ne le fait que ligne à ligne au fur et à mesure de l'exécution.

3. La boucle « pour ».

En C, la syntaxe du « pour » est :

```
for (int i=0; i<10; i++)
{
    // diverses instructions
}
```

On déclare d'abord le compteur `i` comme étant un entier (`int`), qui est créé à la valeur 0. On donne ensuite la condition qui permet d'arrêter la boucle, et enfin on précise la manière dont `i` augmente l'itérateur (`i++` est un raccourci pour `i = i + 1`).

Exercice 1.

La diode doit d'abord clignoter 5 fois une seconde, ceci une seule fois, puis alterner 10 clignotements rapides d'une demi seconde, et 10 clignotements lents de deux secondes sans fin.

Exercice 2.

La diode doit clignoter de plus en plus rapidement, puis de plus en plus lentement, et ainsi de suite. Par exemple en millisecondes : 200 – 400 – 600 – 800 – 1000 – 800 – 600 – 400 – 200 – etc... On peut aussi faire 100 – 200 –

400 – 800 – 1600 – 800 – 400 – 200 – 100 – etc...

Remarque : on peut mettre une variable, ou bien un calcul, dans `delay()`. Les opérations usuelles sont +, -, *, / . Le nom de la variable est au choix, les usages donnent par exemple `tempsEntreDeuxEtats`.

Il faut déclarer cette variable en début de programme (dans le `setup` par exemple).

Exercice 3.

Faire un montage avec deux diodes. Faire clignoter les deux diodes à des rythmes différents (par exemple 210 ms et 980 ms)

Il faudra :

rajouter un compteur de temps général, un pour chaque diode ; on déclare en début de programme (dans le `setup` ou avant) :

```
int minuteur = 0;
int compteurDiode1 = 0 ;
int compteurDiode2 = 0;
```

vérifier ce qui se passe toutes les 10 millisecondes. On utilisera l'instruction conditionnelle « si » :

```
if (condition) { // instructions}
```

les conditions s'écrivent comme pour l'instruction `tant que`.

Pour savoir si le compteur est un multiple des durées de clignotement, on dispose d'une opération importante, le modulo, noté %, qui donne le reste de la division euclidienne d'un nombre par un autre. :

```
x = 2%5 // x contient 2
x = 16%5 // x contient 1
x = 100%5 // x contient 0
```

4. La boucle « tant que ».

En C, la syntaxe du « tant que » est :

```
int i = 0; // déclaration de l'itérateur
while (i < nombre) // ou une autre condition ; exemple i >= nombre
{
    // diverses instructions
    i = i + 1; // ou i++, ou toute autre méthode suivant la condition
    // d'arrêt
}
```

La condition est écrite entre parenthèses, elle peut par exemple être

- `i == 3` // double signe = pour tester l'égalité
- `j <= 4` // inférieur ou égal, de même pour `>`, `<`, `>=`
- `k != 5` // k différent de 5
- `!(k ==5)` // k différent de 5 sous la forme « non k égal à 5 »

Reprendre les exercices du 4 avec une boucle « tant que ».

5. Les capteurs et le port série.

a. *Découverte du port série : copiez le programme suivant.*

```
void setup()
{
  Serial.begin(9600)
}
```

```
void loop() {
  for (int i=1; i<6; i++) {
    Serial.println(200*i);
    delay(200*i);
  }
}
```

Le port série est ouvert avec une vitesse de lecture de 9600 bps (bits par seconde). Considérez-le comme une ligne téléphonique qui peut envoyer ou recevoir jusqu'à 9600 nombres 0/1 par seconde

On « imprime » à l'écran par le port série le nombre `200*i`, avec « `print` ». « `println` » signifie qu'en plus on passe à la ligne après avoir écrit le nombre

Une fois le programme téléversé, il faut cliquer sur le bouton  pour ouvrir la fenêtre du port série. Constatez ce qui se passe.

- Un port série permet la communication de données. Cette communication se fait bit par bit, les bits étant envoyés à la suite les uns des autres. Des protocoles de communications permettent de recomposer l'information envoyée, de vérifier qu'il n'y a pas d'erreur (codes de vérification), le cas échéant de corriger cette erreur (codes de correction) ou bien de redemander l'information. La vitesse à laquelle le port communique est exprimée en baud (nombre de symboles transmis par seconde, qui n'est pas forcément le nombre de bits par seconde, bps). Une fois le programme téléversé dans Arduino, on peut

alors ouvrir le moniteur série avec le bouton . Cela ouvre une fenêtre d'entrée-sortie pour le port série. Dans la fonction setup, on ouvre le port série à 9600 bauds : `Serial.begin(9600)`.

b. *Les senseurs/capteurs*

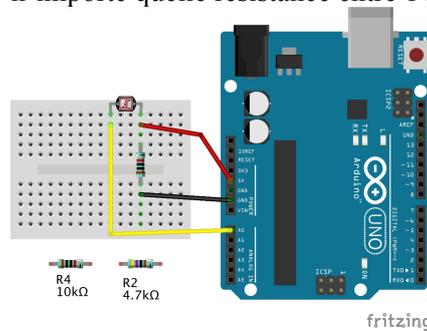
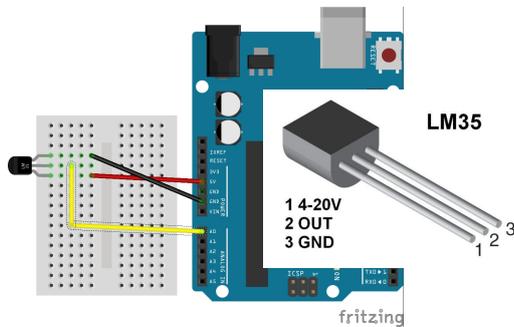
Prenez une thermistance, qui donne la température ambiante. Les senseurs sont des entrées (capteurs) et non des sorties (INPUT et non OUTPUT)

Thermistance LM 35

Renvoie un voltage de 10 mV par degrés. Les port d'entrées analogiques renvoient une valeur correspondante entre 0 et 1023.

Lumistance (ldr)

Appelée CDS parfois. Renvoie un voltage variable suivant la luminosité. Les port d'entrées analogiques renvoient une valeur correspondante entre 0 et 1023. On peut utiliser n'importe quelle résistance entre 1 kΩ et 10 kΩ



```
// en début de programme :
const int inTemp = 0 ; // broche entrée capteur
int valeur ; // lecture de la valeur du LM35
float millivolts = 0 ; // pour la donnée en mV
float celsius ; // conversion en d° Celsius
// dans la boucle principale :
valeur = analogRead(inTemp) // lit la valeur
millivolts = (valeur*5000.0)*1023.0 // conversion
//en mv
celsius = millivolts/10
//conversion en celsius
```

```
// en début de programme :
const int inLum = 0 ; // broche entrée capteur
// dans la boucle principale :
valeur = analogRead(inLum) // lit la valeur
```

Remarque : si vous obtenez des valeurs aberrantes, essayez de changer l'ordre des opérations, par exemple `celsius = (5.0*valeur*100.0)*1023.0`. En effet, rappelez-vous du cours sur le codage : si on dépasse la capacité d'un nombre positif avec un calcul, on se retrouve avec le codage d'un nombre négatif. Gardez toujours les « .0 », cela permet de spécifier que le résultat n'est pas un entier mais un flottant, et évite donc de se retrouver avec 0 comme résultat.

Exercice 4 : écrire dans la fenêtre du port série la donnée résultant du capteur, toutes les secondes.

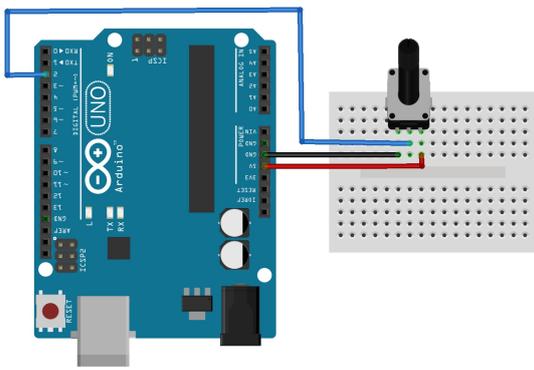
Exercice 5 : Faites l'exercice 1 avec un potentiomètre, qui est une résistance variable. Il modifie la valeur du courant rentrant. Les potentiomètres sont soit à molette, soit à curseur. Attention le code est capricieux et fonctionne plus ou moins bien suivant les années :-).

Montage électronique :

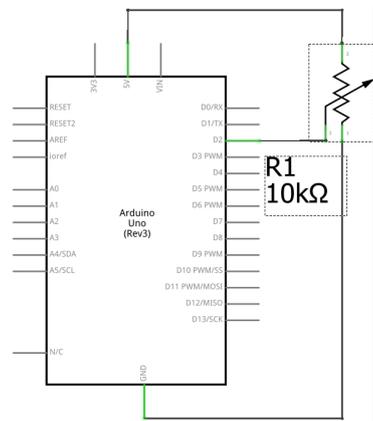
```
const int brochePot = 2 ; //debut de programme
int val = 0 ; // debut de programme

pinMode(brochePot, INPUT) ; // dans le setup

val = analogRead(brochePot) ; // dans la loop
```



fritzing



fritzing

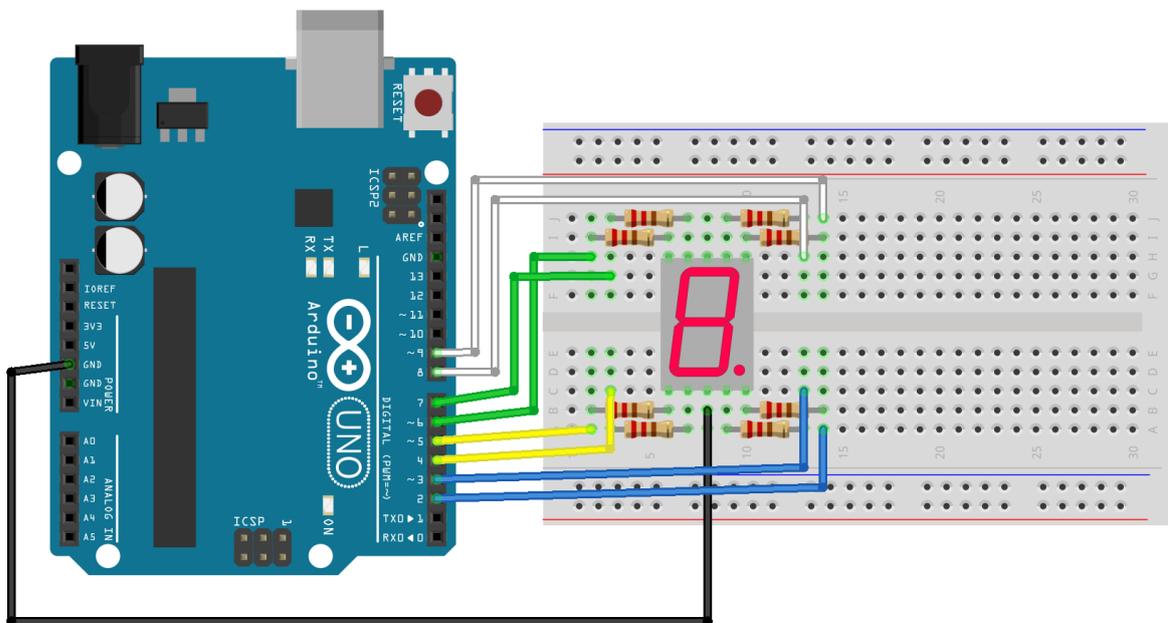
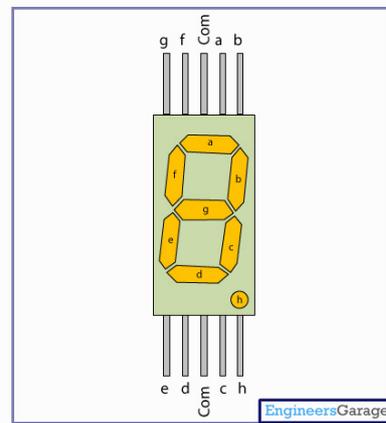
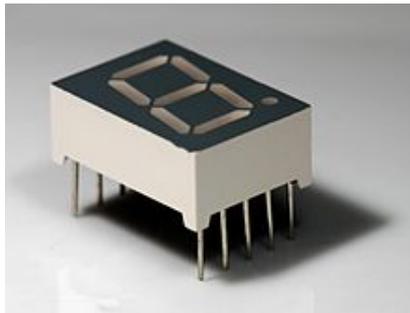


Exercice 6 : Écrivez un programme qui allume une diode avec une luminosité d'autant plus forte qu'il fait plus sombre, avec une lumistance (ou froid, avec une thermistance). On peut réchauffer le thermistor en soufflant dessus.

Exercice 7 : Écrivez un programme qui allume une diode verte s'il fait chaud et une rouge s'il fait froid.

6. Les fonctions ; et les opérations bit à bit.

Réaliser le montage suivant avec le mini-afficheur 7 segments à del. Les résistances sont de 220 ohms minimum (330, 1000, 2000 fonctionnent très bien aussi).



fritzing

Remarque : ce montage est valable pour les afficheurs à cathode commune. S'il ne fonctionne pas, essayez de remplacer le fil noir du ground par un rouge sur le 5V.

Exercice 8 : écrire un programme qui affiche la lettre F, en mettant les « bonnes » sorties à HIGH et les autres à LOW. Le fonctionnement est très semblable à celui des diodes (ce sont des diodes !) ; inspirez-vous des codes des exercices du premier chapitre. Et observez bien les schémas ci-dessus !

Comme vous l'avez vu en faisant l'exercice ci dessus 10, écrire chaque nombre en traitant chaque connexion individuellement est assez long. Il y a plus rapide avec les instructions « bitwise » (bit à bit). Ce type d'instructions permet de traiter les nombres binaires, composés de 8 chiffres qui sont soit 0 soit 1, rapidement. Dans le cas de l'afficheur 8 bits, on traitera les 8 broches avec un seul nombre binaire et peu d'instructions. Un nombre de 8 bits est appelé un octet.

Le programme afficheur_7_segments.ino, téléchargeable sur www.maths-info-lycee.fr/programmes/progr_arduino , et dont le code figure ci-dessous, montre l'utilisation de ces opérations. Récupérez-le.

```
/*représentation des segments allumés par des octets. Chaque octet est
composé de 8 bits : d'abord les 7 segments de l'afficheur, puis le point
décimal. Ces octets sont rangés dans un tableau de 8 éléments, constants
(non modifiables). Le type octet est « byte » en C. */

// création du tableau de 8 bytes/octets. Le tableau s'appelle « segments »
//Un byte commence par B, pour montrer que ce n'est pas un nombre entier standard

const byte segments[8] = {
//ABCDEFGdp // correspondance entre les bits et les segments de l'afficheur
B00000001, // segment A.
B00000010, // segment B
B00000100, // C
B00001000, // D
B00010000, // E
B00100000, // F
B01000000, // G
B10000000, // point décimal (dp)
B00000000, // tout éteint };

// tableau des broches pour chaque segment et le point décimal. C'est un tableau
d'entiers « int » constants
// ordre des broches dans le montage dp2,C3,D4,E5,G6,F7,A8,B9
const int brochesSegments[8] = {8,9,3,4,5,7,6,2}; // vérifier suivant votre câblage
à l'aide des schémas ci dessus

void setup(){
  for(int i=0; i < 8; i++){
    pinMode(brochesSegments[i], OUTPUT); // toutes les broches pour
                                         // l'afficheur sont des sorties
  }
}

void loop(){
  // boucle qui affiche tous les segments un par un, puis éteint tout
  for(int i=0; i <= 8; i++) {
    afficheSegment(segments[i]); // la fonction qui affiche est apres
                                  // le loop, voir ci-dessous
    delay(1000);
  }
}

// Contrôle de l'afficheur. La valeur à afficher est récupérée
// en entrée dans le byte « octet »
void afficheSegment( byte octet){
  // la variable segmentAllumeOuEteint est un booléen,
  // qui prend soit la valeur 1 = True ou la valeur 0 = False.
  boolean segmentAllumeOuEteint;

  // la boucle suivante lit les bits de « octet » 1 par 1,
  // avec le compteur numeroSegment
```

```

for(int numeroSegment = 0; numeroSegment < 8; numeroSegment++){
    // segmentAllumeOuEteint est vrai si le bit « numeroSegment »
    // lu dans l'octet est 1
    segmentAllumeOuEteint = bitRead(octet, numeroSegment);
    // la ligne suivante inverse la valeur de bitAllumeOuEteint. Suivant le
    // type de l'afficheur, il faut la supprimer ou la garder
    //(si les segments sont allumés au lieu d'être éteints)
    // le « ! » est l'opérateur « non », qui inverse vrai et faux.
    // segmentAllumeOuEteint = ! segmentAllumeOuEteint; // ligne facultative
    // pour les afficheurs à anode commune
    digitalWrite( brochesSegments[numeroSegment], segmentAllumeOuEteint);
}
}

```

Exercice 9 : modifier le programme précédent pour qu'il affiche les chiffres de 0 à 9.

Exercice 10 : modifier le programme précédent pour qu'il affiche un chiffre au hasard compris entre 0 et 9. Remarquez que l'on peut aussi écrire les lettres A, b, C, d, E et F. Il y a deux endroits où l'on modifie le programme. D'une part, il faut changer le tableau « segments » ; on peut aussi changer son nom en « chiffres » pour que cela soit plus clair. D'autre part, dans la boucle principale, on rajoutera le choix d'un nombre aléatoire avec l'instruction :

```
float nombreAlea = random(0,9) ; // le 0 est inclus, le 9 est exclu
```

Si l'on veut en plus avoir l'affichage des lettres, il faut écrire `random(0,16)` .

L'instruction `random` génère des nombres pseudo-aléatoires à partir d'une « graine » (seed en anglais). Pour avoir une graine aléatoire, on l'initialise dans le setup en lisant la valeur d'une broche non connectée, qui donne alors du « bruit » analogique aléatoire, avec `randomSeed(analogReag(0))` ; Ici on suppose que l'entrée 0 n'est pas connectée.

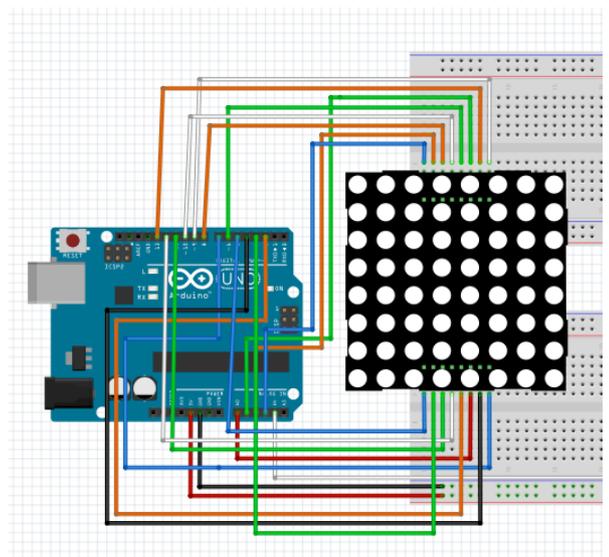
7. Matrice de leds.

Un projet sympa : un snake sur une matrice de leds. Demander le programme au professeur, rien que le câblage est difficile, comme on peut le voir sur le schéma ci-contre (sans aucune logique apparente, du moins pour votre professeur).

8. Les micro-projets avec Tinkerkit.

Tinkerkit permet de faire l'économie des montages électroniques, il suffit de brancher des modules tout faits. Malheureusement, il y a un problème de bibliothèque à résoudre, il faut utiliser une version ancienne de l'IDE Arduino.

Je peux vous fournir un programme « exemple_base.ino », qui est assez idiot dans ses fonctionnalités, mais qui vous montre différents aspects du langage.



Les modules Tinkerkit disponibles sont :

Bouton ; basculement (choc/tilt) ; contact (touch) ; potentiomètre (linéaire ou rotatif) ; photoresistance (light sensor) ; thermistance ; effet hall (mesure du champ magnétique : courant, être humain, aimant, etc... à proximité) ; joystick ; accéléromètre (mesure l'inclinaison/l'accélération) ; gyroscope (mesure l'inclinaison, complémentaire de l'accélormètre) ; DEL (LED) ; relais (permet de commander un second circuit avec un courant bien supérieur, comme du 220 V. Nous ne pourrons nous en servir que pour faire des modèles théoriques, il est interdit de jouer avec des courants au dessus de 24V au lycée !) ; Mosfet (comme le relais, mais à haute vitesse, et avec un courant max de 24V) ; moteurs.

Liens pour les bibliothèques de différents modules si nécessaire : <http://www.tinkerkit.com/tinkerkit-library/> voir en bas de la page oùêbe. Pour utiliser un module, il est nécessaire d'inclure sa bibliothèque en début de programme. On peut dans les exemples voir

Nous disposons aussi de deux shields ethernet (cf <http://arduino.cc/en/Main/ArduinoEthernetShield>) pour connecter Arduino à internet. Réservé aux spécialistes... en effet, on ne passe pas par un navigateur !

Indispensable : l'aide pour le langage : <http://arduino.cc/en/Reference/HomePage>

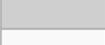
Les mini-projets proposés sont les suivants :

- La boîte à gourmandises qui surveille votre santé : refuse de s'ouvrir pendant un certain temps si elle a été ouverte plus de trois fois dans la dernière demi-heure. Il y a 3 boîtes disponibles, deux ont un capteur de basculement (tilt sensor), une un accéléromètre, qu'il est nécessaire de calibrer avant utilisation (ie tester le fonctionnement pour essayer de le comprendre). Dans un deuxième temps, on peut rajouter un bouton marche/arrêt, ou bien un capteur de contact (à l'arrêt décider d'un comportement constant de la boîte). La version « capteur de contact » est un peu plus difficile à réaliser. Utiliser les fonctions sur le temps (Time...).
- Un thermomètre archi-basique. On l'allume/éteint avec un bouton ou bien un capteur de contact. Une diode verte signale que la température est correcte, une diode rouge qu'elle est trop élevée (et éventuellement une bleue qu'elle est trop basse). La version « capteur de contact » est un peu plus difficile à réaliser.
- Un minuteur qui compte en binaire (sous forme de DEL allumées/éteintes). On pourra utiliser les opérations sur les bits, notamment le `bitRead()`. On pourra le déclencher à partir d'un capteur (toucher par exemple)
- Version alternative : une horloge en binaire. Utiliser les fonctions sur le temps (Time...).
- Une « lampe » composée de plusieurs diodes, dont l'intensité varie en fonction de la luminosité ambiante. On fera varier le nombre de diodes allumées, ainsi que l'intensité de la dernière allumée (les précédentes étant au maximum).
- Un jeu d'adresse : deux accéléromètres/gyroscopes coincés entre deux doigts (par exemple l'un entre index et majeur, l'autre entre annulaire et auriculaire) doivent être maintenues le plus longtemps possible à l'horizontale. Un système de diodes donne : le début du jeu, le score (par exemple de 1 à 3 diodes allumées), la fin du jeu.
- Un mini-projet plus difficile : utiliser un shield ethernet pour aller chercher la météo sur le web, et allumer des diodes en fonction du temps qu'il va faire (cf livre arduino cookbook pour un peu d'aide).

Annexe 1 : lecture de la valeur d'une résistance



Code des couleurs pour les résistances

	1 ^{er} anneau gauche	2 ^e anneau gauche	3 ^e anneau gauche*	Dernier anneau gauche	Anneau droite	Anneau suppl.	Abrév.
Couleur	1 ^{er} chiffre	2 ^e chiffre	3 ^e chiffre	Multiplicateur	Tolérance	Coeff. temp.	Alpha.
 noir	0	0	0	10 ⁰ =1	± 20 %	200 ppm	BK
 marron	1	1	1	10 ¹	± 1 %	100 ppm	BN
 rouge	2	2	2	10 ²	± 2 %	50 ppm	RD
 orange	3	3	3	10 ³		15 ppm	OG
 jaune	4	4	4	10 ⁴		25 ppm	YW
 vert	5	5	5	10 ⁵	± 0,5 %		GN
 bleu	6	6	6	10 ⁶	± 0,25 %		BU
 violet	7	7	7	10 ⁷	± 0,10 %		VT
 gris	8	8	8	10 ⁸	± 0,05 %		GY
 blanc	9	9	9	10 ⁹			WT
 or				0,1	± 5 %		GD
 argent				0,01	± 10 %		SR
 (absent)					± 20 %		

* Le troisième anneau n'est utilisé que lorsque la tolérance de la résistance est inférieure à 2 %.

Un moyen mnémotechnique pour se souvenir de l'ordre des couleurs dans le code couleur des résistances est de connaître la phrase suivante : "Ne Manger Rien Ou Jeûner Voilà Bien Votre Grande Bêtise" ou encore "Ne Mangez Rien Ou Je Vous Brûle Votre Grande Barbe" (dans les deux cas, vert est avant violet, comme dans le dictionnaire). Chaque initiale correspond à la première lettre de chaque couleur.