

# Introduction à la robotique avec Arduino

## 1. Introduction

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

Deux versions sont proposées, l'une avec de l'électronique, l'autre sans. Pour ceux qui ont peur de l'électronique, vous commencerez avec les modules « Tinkerkit », qui évitent de faire des montages. Vous ferez ensuite, si vous en avez le temps, la version « électronique pure ». Les modules Tinkerkit sont des briques, que l'on connecte tout simplement par des câbles.

Je vous demanderai de faire preuve de mesure et de délicatesse lors des manipulations, les composants sont fragiles. Vous travaillerez à deux. 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 électroniques (kit de débutants pour Arduino par exemple) dans les 40 euros (on peut trouver moins cher encore, la qualité/solidité sera en conséquence). Et si vous n'avez pas peur de l'anglais, un très bon bouquin, complet : Arduino Cookbook, Michael Margolis, éditions O'Reilly (il existe aussi des livres en français, plus simples et très bien faits, ainsi que des tutoriels sur le net, de qualité variable). Vous pouvez aussi utiliser des modules comme TinkerKit, c'est un peu plus cher et un peu plus simple. Dans ce cas je vous conseille plutôt les modules « Grove », Tinkerkit étant en sommeil ces dernières années.

## 2. Premier programme.

*Matériel* : vous aurez besoin uniquement d'une carte Arduino et d'un câble USB

Nous allons faire clignoter une des LED (light emitting diode, diode électro luminescente ou DEL en français) située sur la carte Arduino, sans faire de montage pour l'instant. C'est la diode marquée L sur la carte (repérez-la).

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 :

Programme	Explications
// Programme écrit par Genie_1 et Genie_2	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 */
void setup(){	Le « setup » est l'initialisation du programme, qui ne sera exécuté qu'une seule fois au démarrage. On ouvre le setup avec une accolade
pinMode(13,OUTPUT);	On déclare la connexion 13 comme étant une sortie de courant. Chaque ligne de programme se finit par un point-virgule. On décale les instructions, d'une tabulation ou de quatre espaces, à chaque fois que l'on ouvre des nouvelles accolades.
}	Fermeture du setup
void loop() {	Boucle principale, exécutée à l'infini
digitalWrite(13,HIGH);	On envoie du courant dans la broche 13.
delay(1000);	On attend 1 seconde (1000 millisecondes)
digitalWrite(13,LOW);	On coupe le courant dans la broche 13.
delay(1000);	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.)

### 3. Premier montage.

#### a. Version tinkerkit

*Matériel* : une carte Tinkerkit (un « shield »), un câble Tinkerkit et un module DEL. Branchez le shield sur l'Arduino, et la diode sur le port O0 (Output 0).

Le programme précédent est transformé comme ceci :

```
#include <TinkerKit.h>      //la bibliotheque TinkerKit permet d'utiliser les
                             //modules

TKLed led(O0);              // declaration de la DEL en sortie
void setup() {
}

void loop() {
  led.on();                  // on allume la DEL
  delay(1000);
  led.off();                 // on eteint la DEL
  delay(1000);
}
```

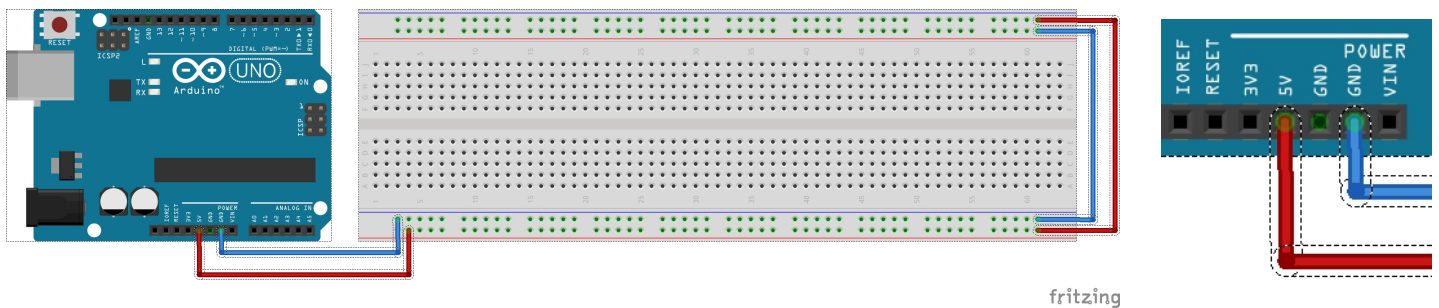
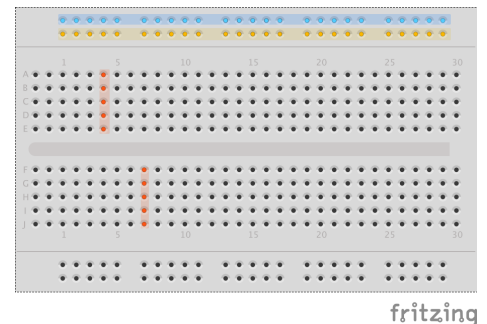
#### b. Version montage électronique

*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

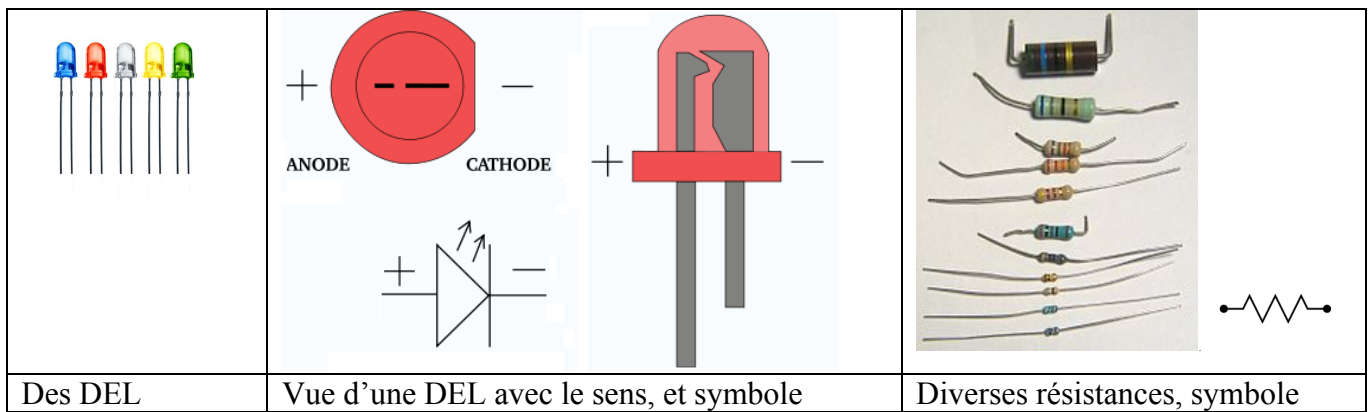
Les trous d'en haut/bas servent souvent pour le courant. Par convention, on met un fil rouge/marron pour l'entrée, un fil bleu pour la sortie et un fil noir/bicolore pour la terre pour le 220 V. La couleur rouge est impérativement respectée car dangereuse, notamment avec le 220 V : ne touchez **jamais**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

Ci dessous la connexion du courant sur la breadbord, avec un agrandissement du branchement sur la carte.



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).



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 (rouge rouge marron, vérifier avec la table/méthode ci-dessous). Plus la résistance est forte moins la diode émettra de lumière.



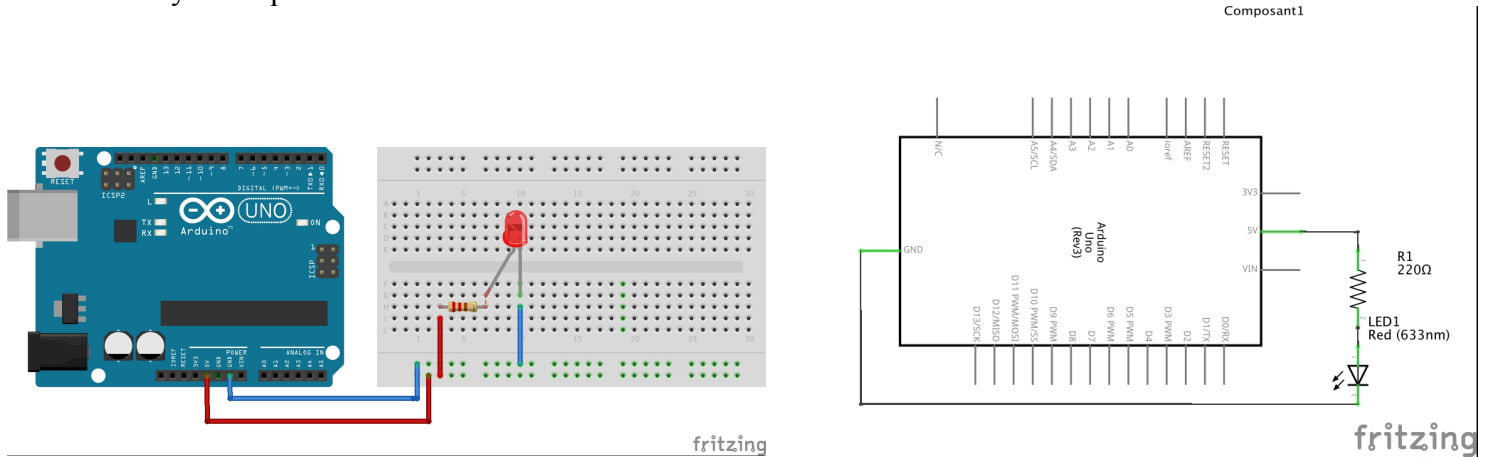
Lecture de la valeur d'une résistance :

**Code des couleurs pour les résistances**

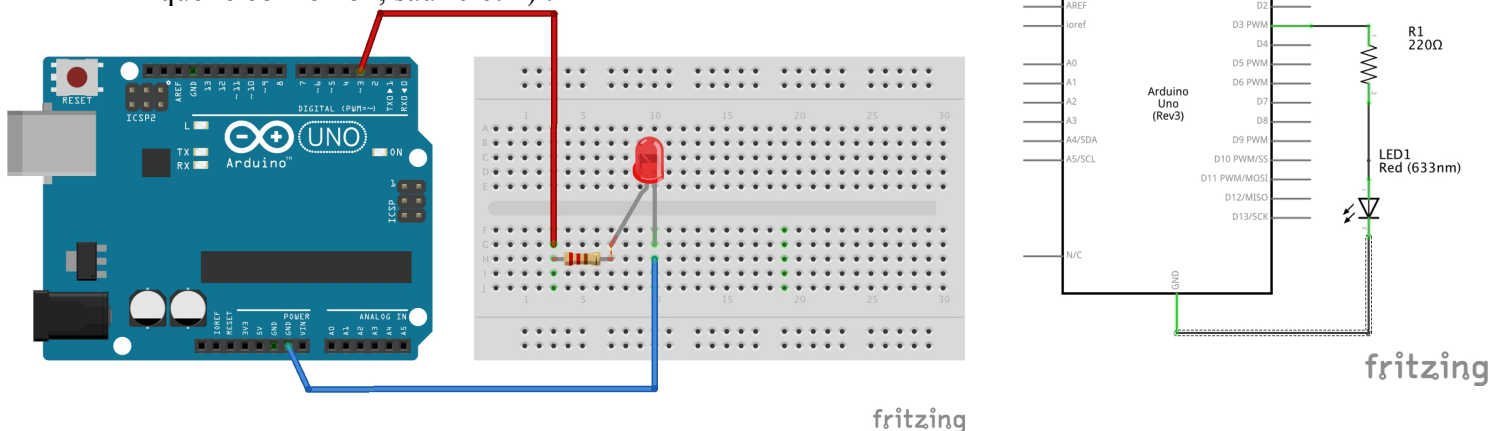
	1 <sup>er</sup> anneau gauche	2 <sup>e</sup> anneau gauche	3 <sup>e</sup> anneau gauche*	Dernier anneau gauche	Anneau droite	Anneau suppl.	Abrév.
Couleur	1 <sup>er</sup> chiffre	2 <sup>e</sup> chiffre	3 <sup>e</sup> chiffre	Multiplicateur	Tolérance	Coeff. temp.	Alpha.
noir	0	0	0	10 <sup>0</sup> =1	± 20 %	200 ppm	BK
marron	1	1	1	10 <sup>1</sup>	± 1 %	100 ppm	BN
rouge	2	2	2	10 <sup>2</sup>	± 2 %	50 ppm	RD
orange	3	3	3	10 <sup>3</sup>		15 ppm	OG
jaune	4	4	4	10 <sup>4</sup>		25 ppm	YW
vert	5	5	5	10 <sup>5</sup>	± 0,5 %		GN
bleu	6	6	6	10 <sup>6</sup>	± 0,25 %		BU
violet	7	7	7	10 <sup>7</sup>	± 0,10 %		VT
gris	8	8	8	10 <sup>8</sup>	± 0,05 %		GY
blanc	9	9	9	10 <sup>9</sup>			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 Jeuner 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.

Voici une première proposition de montage, à ne pas faire, avec à côté le schéma électronique symbolique :



En fait, nous allons utiliser les broches (pin en anglais) 0 à 13 pour brancher les composants. En effet, le montage ci-dessus envoie du courant en permanence, tandis que l'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) :



Reprendre le programme du 1 et le tester sur votre montage. **Attention** : déclarer la bonne connexion, ce n'est plus forcément la 13.

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

#### 4. La boucle « pour ».

La boucle « pour i de 0 à xxx » est une instruction qui répète plusieurs fois un bloc d'instructions, en général xxx fois. On dit que c'est une instruction itérative (une itération = une répétition). On utilise un compteur de boucle (itérateur) en général i, j ou k. On précise de combien on augmente le compteur de boucle à chaque itération.

*Exemples* :

- $i = i + 1$       augmente de 1 le compteur i à chaque itération (standard) ;
- $i++$             augmente de 1 le compteur i à chaque itération (raccourci pour  $i = i + 1$ ) ;
- $i = i - 1$       diminue de 1 le compteur à chaque itération ;
- $i = i * 2$         multiplie par 2 le compteur à chaque itération.

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 (on arrête dès que  $i \geq 10$ ), et enfin on précise la manière dont `i` augmente l'itérateur.

5. Exercices à faire avec le montage ci-dessus (que ce soit en version Tinkercat ou électronique).

Tous les exercices seront recopiés dans le dossier `documents>devoirs>Mandon`. Ils seront nommés comme suit : `ex_arduino_1_1_Nom1_Nom2.ino` (pour le 1<sup>er</sup> exercice ci-dessous, exercice 1\_1, puis vos noms)

a. Exercice 1\_1.

Faire clignoter la diode alternativement deux secondes, puis une seconde (remarque : pas besoin de « for »).

b. Exercice 1\_2.

La diode doit clignoter 10 fois une seconde, puis s'arrêter.

c. Exercice 1\_3.

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.

d. Exercice 1\_4.

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).

6. La boucle « tant que ».

La boucle « tant que/while » est aussi une instruction itérative. Elle répète un bloc d'instruction suivant une condition.

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 »

La différence entre « pour » et « tant que » est que « pour » exécute un nombre de fois prédéterminé le bloc, tandis que le nombre d'itérations du « tant que » n'est pas déterminé à l'avance.

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
}
```

*Reprendre les exercices du 4 avec une boucle « tant que ». Vous les numéroterez 1\_1b, etc.*

Ce cours est sous licence Creative Commons BY NC SA, voir <http://creativecommons.org/licenses/by-nc-sa/3.0/fr/>, il a été écrit par Frédéric Mandon. Merci à Wikipédia pour les images et les codes des résistances (entre autres), au cours Open Classrooms « programmez vos premiers montages avec Arduino », notamment pour les explications claires sur les calculs des valeurs des résistances. Tous les schémas de montage ont été faits sur le logiciel Fritzing.

## Complément : calcul des résistances

Dans un circuit électrique, on note  $U$  (ou  $V$ ) la tension (ou différence de potentiel) en volts,  $I$  l'intensité, en ampères, et  $R$  la résistance, en ohms. On peut comparer le courant électrique au flot d'une rivière. La tension correspond alors à la différence d'altitude entre la source et l'embouchure : plus la différence est importante, plus le flot coulera fort : un ruisseau de montagne a une « différence de potentiel » plus forte que la Seine. L'intensité correspond au débit du flot d'eau : la Seine a une « intensité » plus grande qu'un ruisseau de montagne.

La puissance, en Watts, est le produit  $U \cdot I$  :

Les différences de potentiel nous entourant vont de 20/90 mV (nerfs optiques/musculaires), à 1 000 000 V (la foudre), en passant par 1,5 V (les piles), 220 V (les prises de courant), 10 000/500 000 V (lignes à haute tension).

On utilise trois lois pour calculer les résistances à utiliser dans un circuit électrique :


- la loi des mailles
- la loi des nœuds
- la loi d'Ohm :  $U = RI$

# Introduction à la robotique avec Arduino - 2

## 1. Les capteurs et le port série.

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

<pre>void setup() {   Serial.begin(9600) }  void loop() {   for (int i=1; i&lt;6; i++) {     Serial.println(200*i);     delay(200*i);   } }</pre>	<p>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</p>
<pre>void loop() {   for (int i=1; i&lt;6; i++) {     Serial.println(200*i);     delay(200*i);   } }</pre>	<p>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</p>

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

### b. Les senseurs/capteurs

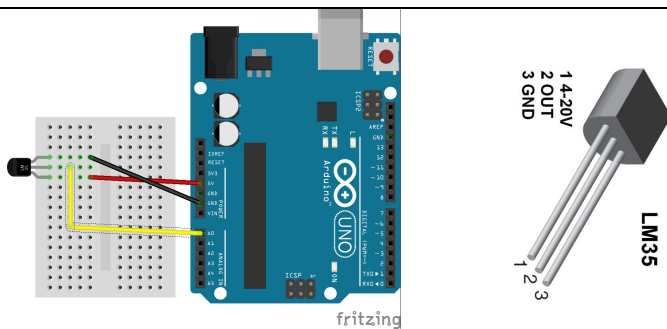
Prenez soit une photorésistance (ou encore light sensor), qui capte la luminosité ambiante, soit une thermistance, qui donne la température ambiante. Les senseurs sont des entrées et non des sorties (INPUT et non OUTPUT)

Avec TinkerKit : branchez le capteur sur l'entrée IO (input 0)

Instructions	Lumistance	Thermistance
Déclaration en entrée 0 (input 0), à écrire après l'importation de la bibliothèque Tinkerkit	<code>TKLightSensor ldr(IO);</code>	<code>TKThermistor therm(IO);</code>
Lecture de la valeur (luminosité ou température)	<pre>int luminosite; // variable à déclarer au début du programme (avant le setup) luminosite = ldr.read(); // la valeur lue va de 0 à 1023</pre>	<pre>int valeur ; float Celsius ; // variables à déclarer au début du programme (avant le setup) valeur = therm.get() ; // la valeur lue va de 0 à 1023 Celsius = therm.getCelsius() ; // la valeur lue est ici donnée en degrés celsius</pre>

### Thermistance LM 35

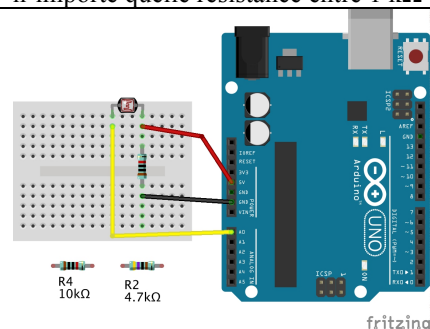
Renvoie un voltage de 10 mV par degrés. Les port d'entrées analogiques renvoient une valeur correspondante entre 0 et 1023.



```
// 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/1023.0)*5000.0 //
conversion
//en mv
celsius = millivolts/10 //conversion en celsius
```

### 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 inLum = 0 ; // broche entrée capteur

// dans la boucle principale :
valeur = analogRead(inLum) // lit la valeur
```

Exercice 2\_1 : écrire dans la fenêtre du port série la donnée résultant du capteur, toutes les secondes.

*Exercice 2\_2b* : 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.

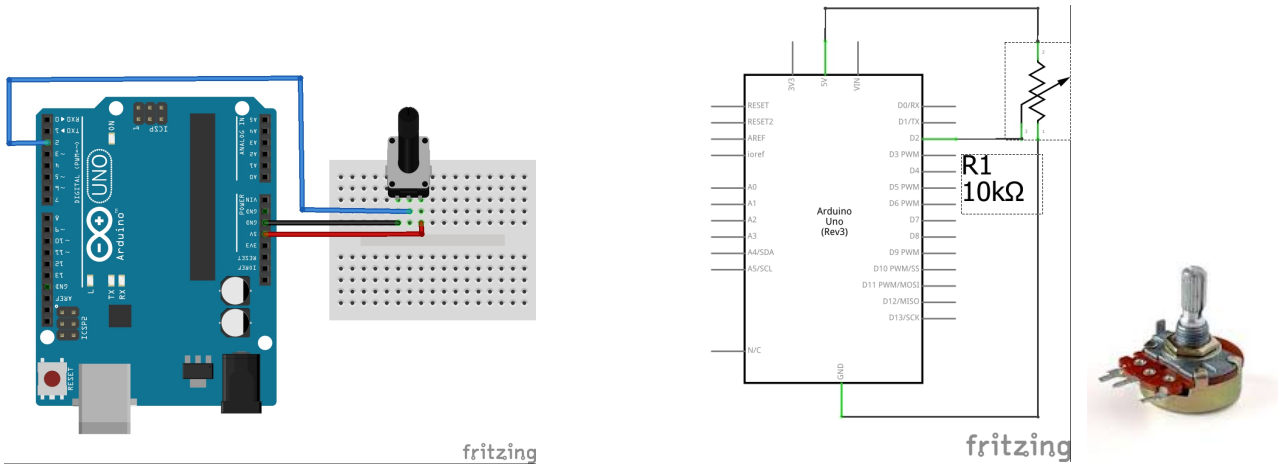
*TinkerKit* : déclaration `TKPotentiometer pot(I0)`; et lecture `valeur = pot.read()`;

*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
```



*Exercice 2\_2* : Écrivez un programme qui allume une diode avec une luminosité d'autant plus forte qu'il fait plus sombre (ou froid). On peut réchauffer le thermistor en soufflant dessus.

*Tinkerkit* : la luminosité d'une diode se règle avec `led.brightness(val)`; où `val` est une variable déjà récupérée précédemment (et déclarée en début de programme).

## 2. L'instruction conditionnelle si/sinon.

*Exercice 2\_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`

## 3. Un simple bouton.

Il y a deux types d'entrées et de sorties :

- les entrées/sorties logiques, qui ont deux états LOW/HIGH, ou bien 0/1
- les entrées/sorties analogiques, qui ont une plage de valeurs. Sous arduino, c'est de 0 à  $1023 = 2^{10} - 1$



On va travailler sur les boutons pour pouvoir donner des ordres simples à la platine Arduino.

Un bouton est une entrée logique (INPUT).

*TinkerKit :*

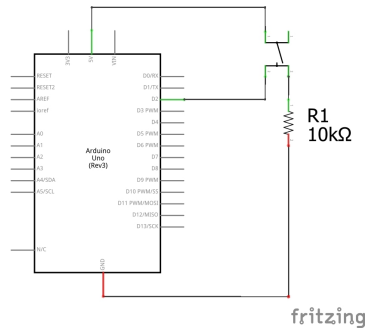
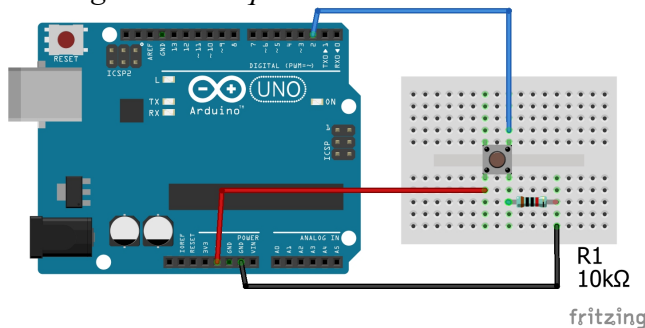
```
TKButton bouton(I0); // déclaration du bouton (en debut de programme)
val = bouton.read() ; // lecture de la valeur : HIGH si le bouton est
                        // pressé, LOW sinon (dans le loop)
```

*Remarque :* sous Tinkerkit, on peut mettre un capteur de contact à la place d'un bouton.

Un capteur de contact fonctionne comme un bouton tactile, il suffit de le toucher. Par contre, comme les mesures se font très rapidement, en quelques millisecondes, il peut être utile de rajouter un délai de lecture entre deux lectures de l'état.

```
TKTouchSensor touch(I0); // déclaration du bouton
val = touch.read() ; // lecture de la valeur : HIGH si le bouton est
                    // pressé, LOW sinon
val = touch.readSwitch() ; // lecture de la valeur comme sur un
                            // interrupteur : l'appui change la valeur de
                            // HIGH en LOW et vice-versa
```

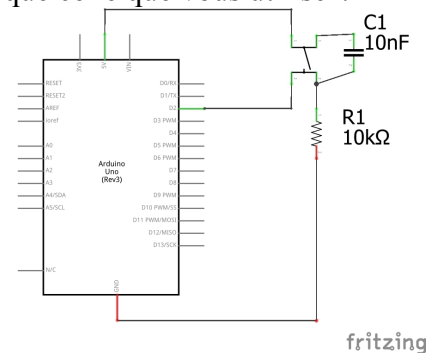
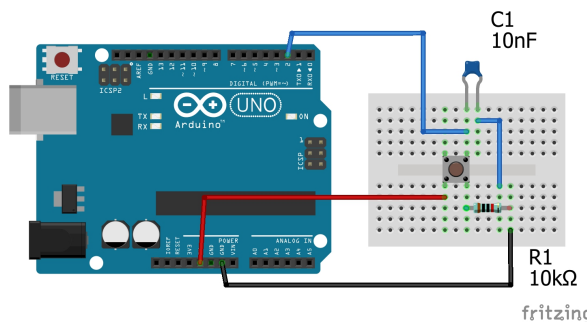
*Montage électronique :*



Quand le bouton est ouvert (non pressé), il n'y a pas de connexion entre les deux broches du boutons, et la connexion est au sol via la résistance. On lit donc « low » sur la connexion 2. Quand on appuie, le courant passe entre les deux broches, et on lit « high ».

Remarques :

- une broche d'entrée-sortie qui n'est connectée à rien fluctue de manière aléatoire entre Haut et Bas. C'est pour cela que l'on met la résistance « pull-down » (rappel vers le moins, vers le ground).
- On peut monter ce circuit à l'envers, avec la résistance en « pull-up » (rappel vers le plus) plutôt qu'en « pull-down ». Le bouton fonctionnera alors à l'opposé. Arduino possède en interne une résistance « pull-up ». SI on l'utilise, on ne met pas de résistance sur le montage et on écrit dans le programme `pinMode(brocheBouton, INPUT_PULLUP)` ;
- On peut aussi rajouter un condensateur en parallèle sur le bouton. Ce composant amortit les rebonds du courant. En effet, lorsqu'on appuie sur le bouton, le courant ne passe pas immédiatement d'un état à l'autre, il oscille avant de se stabiliser. Si vous voulez faire le montage avec condensateur, il faudra prendre un condensateur de 10 nF (nano Farad), qui est dans une autre boîte que celle que vous utilisez.



*Exercice 2\_4 :* afficher l'état du bouton à l'aide du port série

*Exercice 2\_5 :* Reprendre le programme de l'exercice chap2\_1\_3. Rajouter un bouton qui met en marche ce programme, ou l'éteint.

#### 4. « Mappage ».

L'instruction `map` permet de transformer une plage de valeurs en une autre (`map` = carte).

Exemple :

```
pourcentage = map(valeur, 0, 1023, 0, 100) // transforme valeur, comprise entre 0 et 1023, en un
pourcentage compris entre 0 et 100, grâce à une fonction affine (au passage un exercice de mathématiques :
donner l'expression de la fonction affine en question). On peut mettre des variables plutôt que des nombres.
```

*Exercice 2\_6* : faire clignoter une led plus ou moins vite, suivant la position d'un potentiomètre, grâce à un mappage.

*Exercice 2\_7* : Reprendre le programme de l'exercice chap2\_3\_2.

On va reprendre ce programme en « mappant » la donnée du capteur (`map` signifie carte en anglais). Le but est d'avoir une luminosité maximale/minimale de la diode en fonction des conditions du moment, qui sont variables.

Dans l'initialisation, on va mesurer la donnée maximale, ainsi que la donnée minimale du capteur. Pour cela, il faudra bien sûr déclarer les deux variables en question. On rajoutera également un bouton pour permettre à l'utilisateur d'interagir avec Arduino.

#### 5. Complément : les interruptions.

Une interruption permet d'arrêter le programme en cours pour exécuter des instructions que l'on estime urgentes.

Exemple :

```
// dans le setup par exemple
attachInterrupt(0, Reagir, FALLING);

// en dehors du setup et du loop
Reagir() {
// instructions
}
```

La connexion pour une interruption est uniquement sur les broches 2 et 3.

Les paramètres de « `attachInterrupt` » sont dans l'ordre :

- 0/1 broche 2 ou broche 3 ;
- Nom de la fonction appelée lors de l'interruption
- LOW/FALLING/RISING/CHANGE : type d'événement déclenchant l'interruption, respectivement passage à l'état bas de la broche/passage du haut au bas/passage du bas au haut/changement d'état

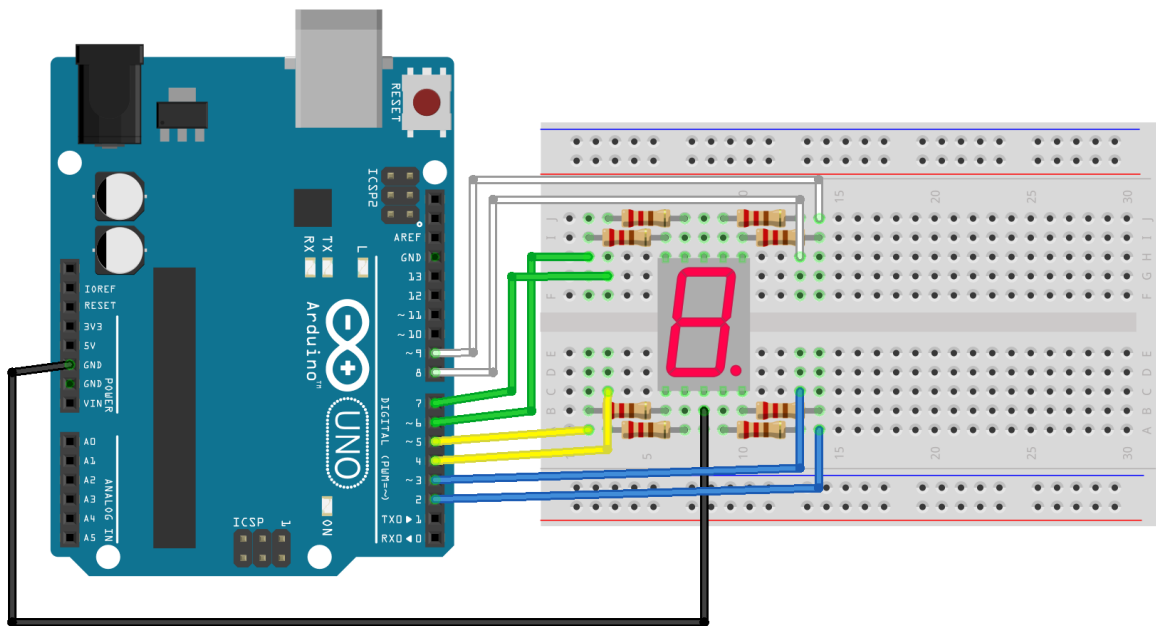
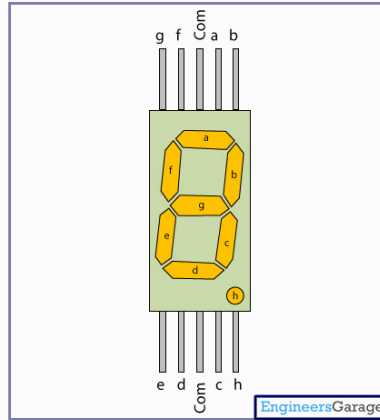
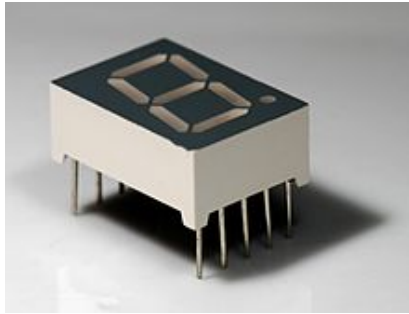
*Exercice 2\_8* : Reprendre le programme de l'exercice chap2\_4\_2. Permettre de mesurer les conditions ambiantes non pas uniquement à l'initialisation du programme, mais tout le temps grâce à une interruption.

## Introduction à la robotique avec Arduino – 3

Les montages de ce chapitre ne peuvent pas se faire avec Tinkerkit.

1. 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 3\_1 : 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 chap3\_1\_1, 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 (dans Documents>Devoirs>Mandon), 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
ranges 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[9] = {
//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 3\_2* : modifier le programme précédent pour qu'il affiche les chiffres de 0 à 9.

*Exercice 3\_3* : 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(analogRead(0))` ; Ici on suppose que l'entrée 0 n'est pas connectée.

---

Ce cours est sous licence Creative Commons BY NC SA, voir <http://creativecommons.org/licenses/by-nc-sa/3.0/fr/>, il a été écrit par Frédéric Mandon. Merci à Wikipédia pour les images et les codes des résistances (entre autres), au cours Open Classrooms « programmez vos premiers montages avec Arduino », notamment pour les explications claires sur les calculs des valeurs des résistances. Tous les schémas de montage ont été faits sur le logiciel Fritzing.

---