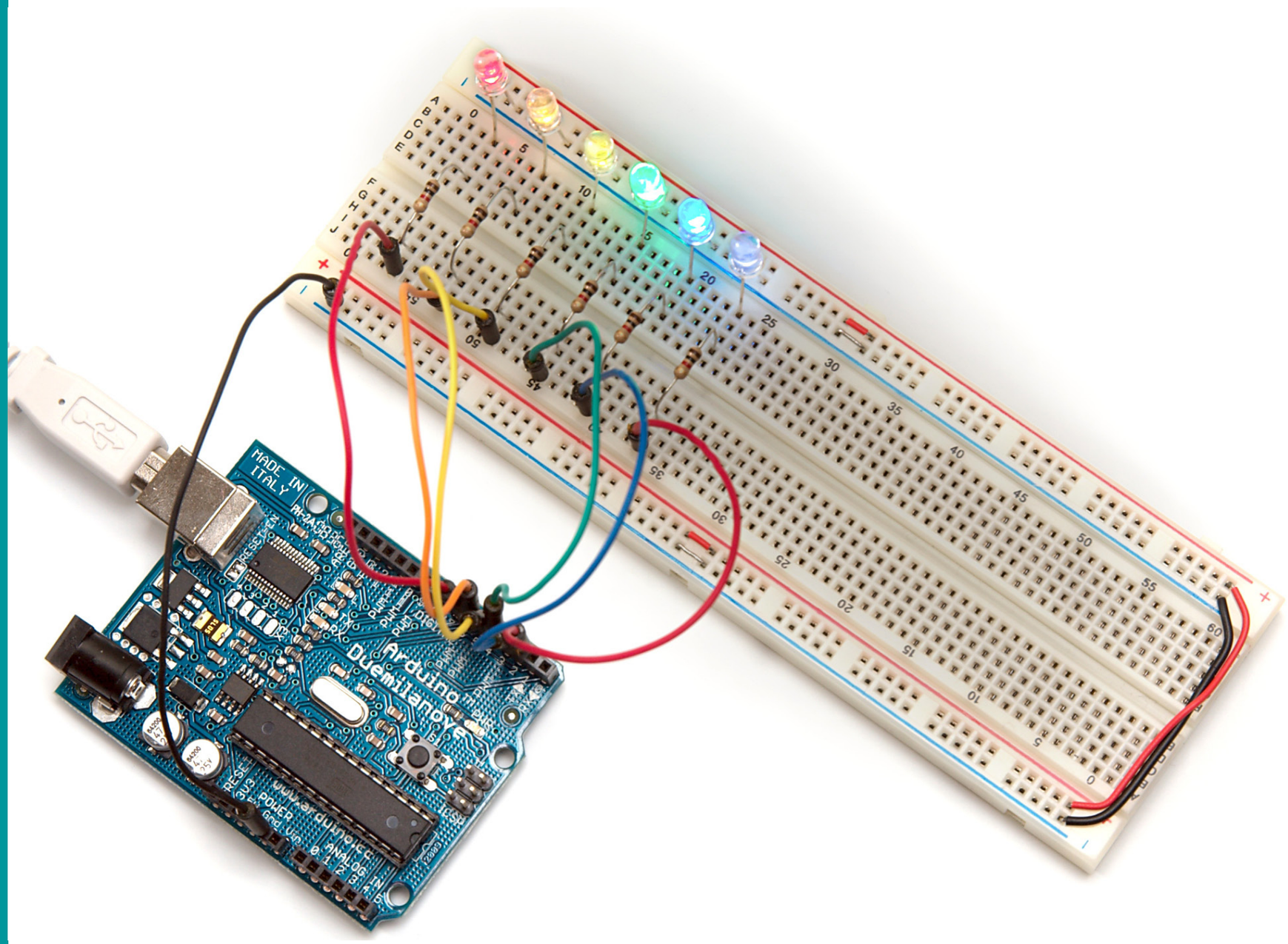
ARDUINO : OUTPUT



Output

Nous allons d’abord nous intéresser à ce qu’on peut faire sortir de Arduino.

Nous allons essayer de faire sortir
- De la lumière
- Du son
- De quoi alimenter un moteur DC
- De quoi piloter un servo motor

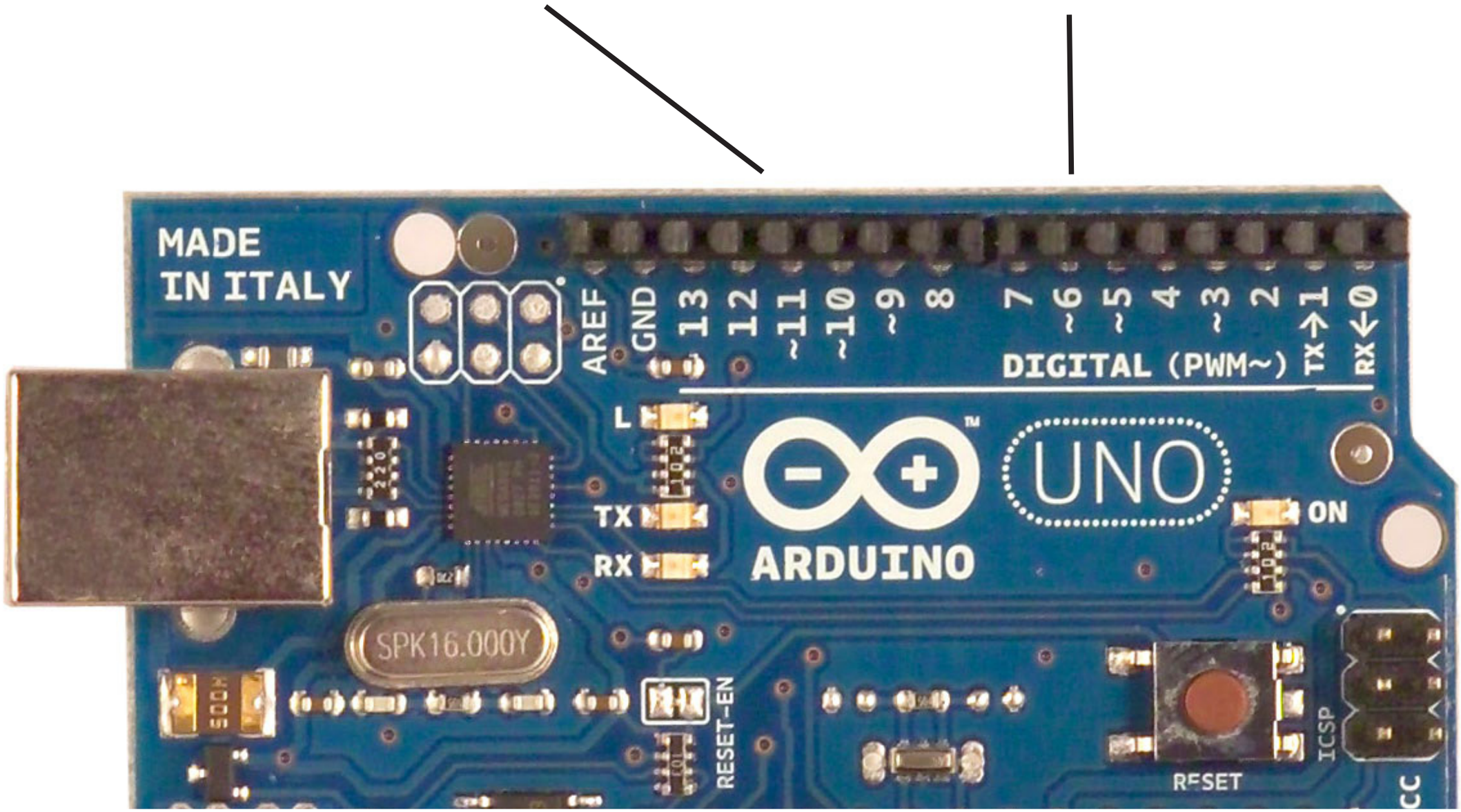
C’est plus facile à comprendre et maitriser.

Cela va nous permettre d’entrer en douceur dans les concepts de programmation et d’électronique.

Rappel : les sorties

Des entrées /sorties digitales

Quelques une sont modulées (PWM)



Entrées ou sorties digitales

Nous avons affaire ici à des entrées/sorties, ce qui signifie que l'on peut les utiliser pour envoyer du courant, ou tester si du courant y entre.

C'est la programmation qui déterminera si Arduino doit les utiliser d'une façon ou d'une autre.

Nous allons voir ça assez vite.

Sorties «digitales»

Digital signifie dans ce cas que ces sorties ne connaissent que deux possibilités :

1 ou 0

HIGH ou LOW

On ou off

5 volts ou 0 volts

(toutes ces oppositions sont équivalentes)

Programmer une sortie

Nous allons concevoir un premier programme.

Il faut dans celui-ci

1) Choisir une des «pin» et spécifier qu'on va l'utiliser comme sortie
2) Envoyer de l'électricité sur cette Pin (et allumer une diode branchée dessus)
3) Attendre un peu
4) Couper l'électricité sur cette Pin
5) Attendre un peu

Facile !

setup() et loop()

Une programmation en Arduino est basiquement constituée par deux fonctions (on reviendra sur la notion de fonction) comme ceci :

```
void setup(){

}

void loop(){

}
```

Retenez que setup() s'exécute une seule fois, et que loop, comme son nom l'indique, se répète tant que Arduino est alimenté.

Assigner la sortie

Nous allons maintenant déterminer la pin 13 comme sortie :

```
void setup(){
    pinMode(13,OUTPUT);
}

void loop(){

}
```

Logiquement, cette opération se fait une seule fois, dans le setup du programme.

Allumer la led

Lors de l'exécution du programme (la loop), envoyons de l'électricité vers la pin 13 :

```
void setup(){
    pinMode(13,OUTPUT);
}

void loop(){
    digitalWrite(13,HIGH);
}
```

HIGH est la même chose que 1 ou que 5 volts.

Attendre !

Attendons une seconde :

```
void setup(){
    pinMode(13,OUTPUT);
}

void loop(){
    digitalWrite(13,HIGH);
    delay(1000);
}
```

delay est une fonction qui fait attendre le programme. La valeur entre parenthèses s'exprime en millièmes de secondes.

Eteindre et attendre

On va maintenant éteindre et attendre une seconde :

```
void setup(){
    pinMode(13,OUTPUT);
}

void loop(){
    digitalWrite(13,HIGH);
    delay(1000);
    digitalWrite(13,LOW);
    delay(1000);
}
```

LOW est la même chose que 0 volts ou off.

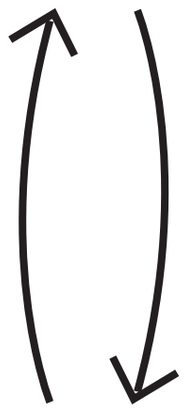
Récapitulatif

On a donc un programme simple mais complet :

```
void setup(){
    pinMode(13,OUTPUT);
}
```

Le setup s'exécute une fois, il contient les paramètres de base du programme

```
void loop(){
    digitalWrite(13,HIGH);
    delay(1000);
    digitalWrite(13,LOW);
    delay(1000);
}
```

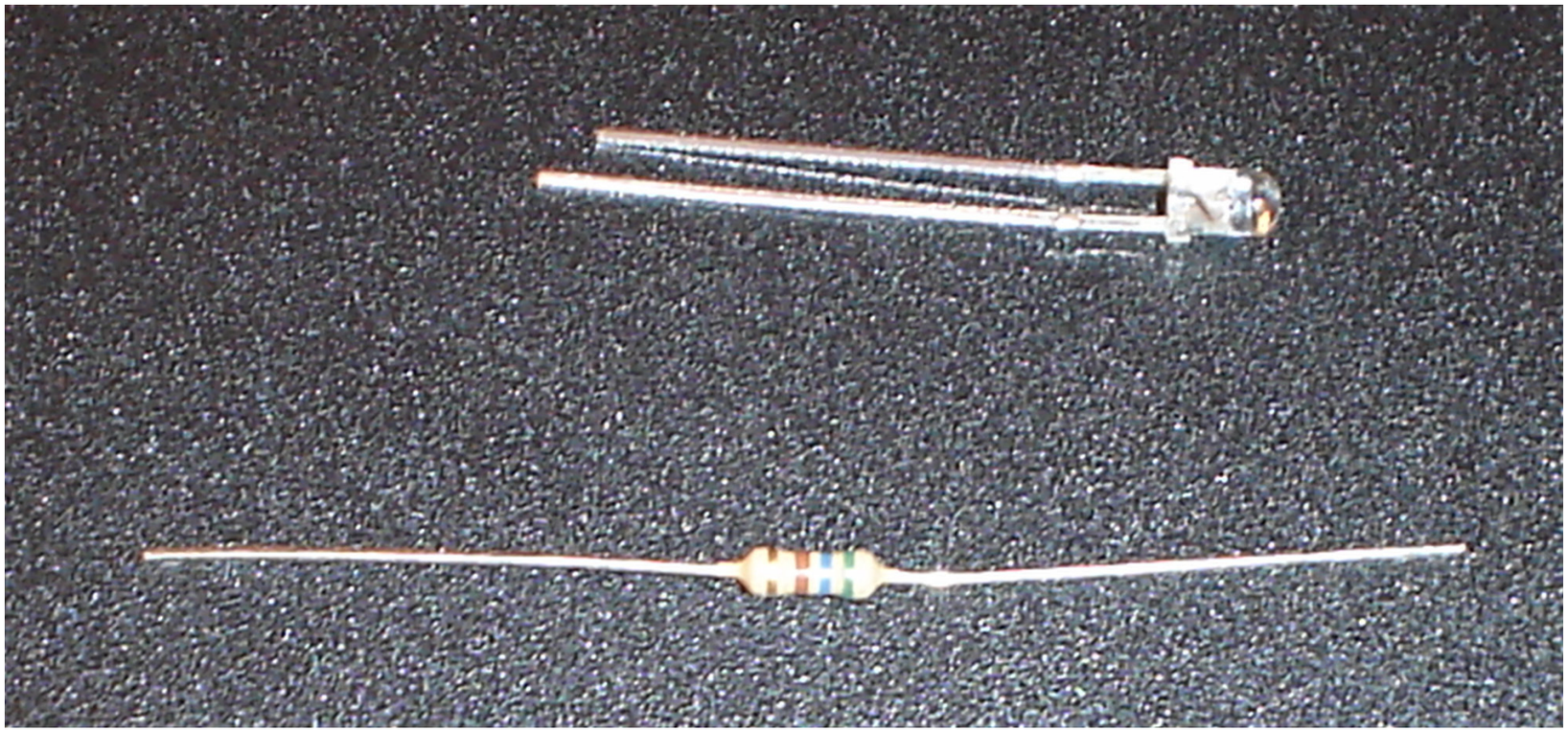


La loop s'exécute en continu. Arrivée à la fin, elle repart au début sans passer par la case «setup()».

Pin 13 et résistance

La pin 13 est un peu particulière : elle possède une résistance (un composant électronique) qui diminue la puissance de sa sortie. C'est ce qui fait qu'on l'a employée pour la démo : un led ne supporte pas autant de voltage.

Nous allons faire un premier exercice qui va nous obliger à utiliser un premier composant électronique : la résistance.

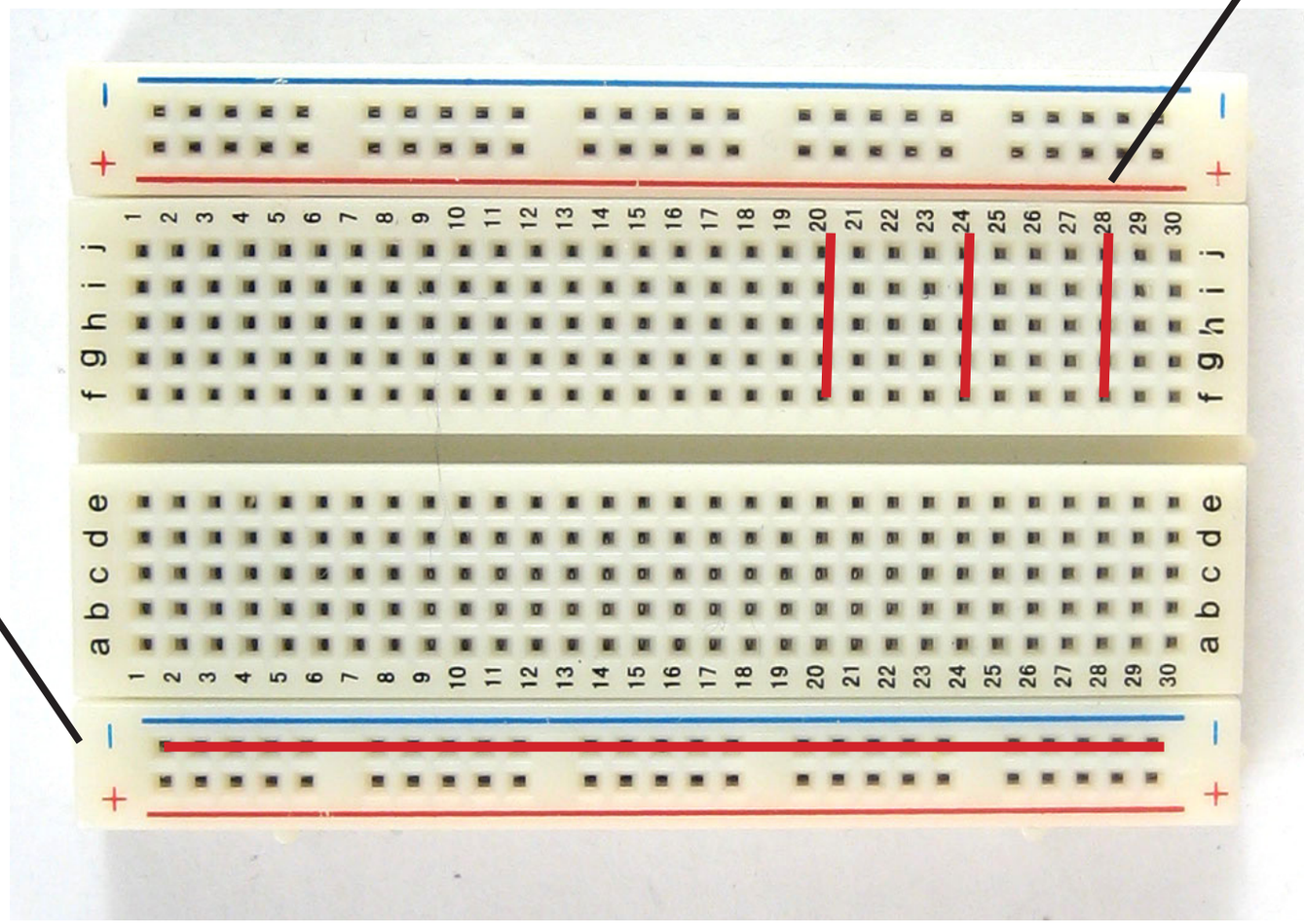


Breadboard

Le breadboard permet de prototyper rapidement.

Au centre, les points sont connectés comme ceci

Sur le côté, les points sont connectés comme ceci

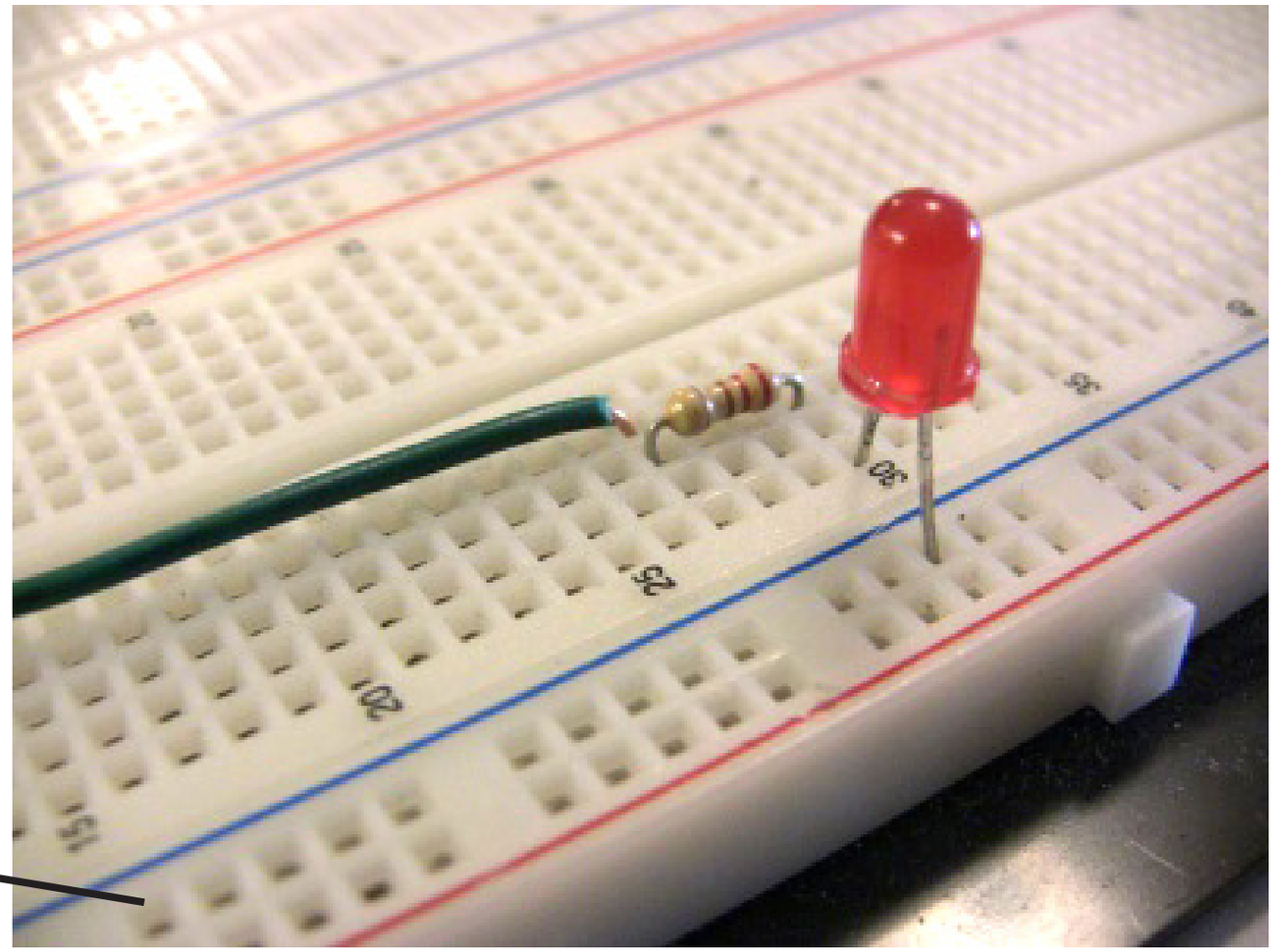


Breadboard : exemple

Pour placer une led, on procedera typiquement comme ceci :

ce fil va vers la pin

On connecte le ground sur la ligne bleue, pour l'employer plusieurs fois.



Un feu de signalisation

Pour ce premier exercice, il s'agit d'allumer et éteindre 3 leds de manière cyclique.

Rappel :

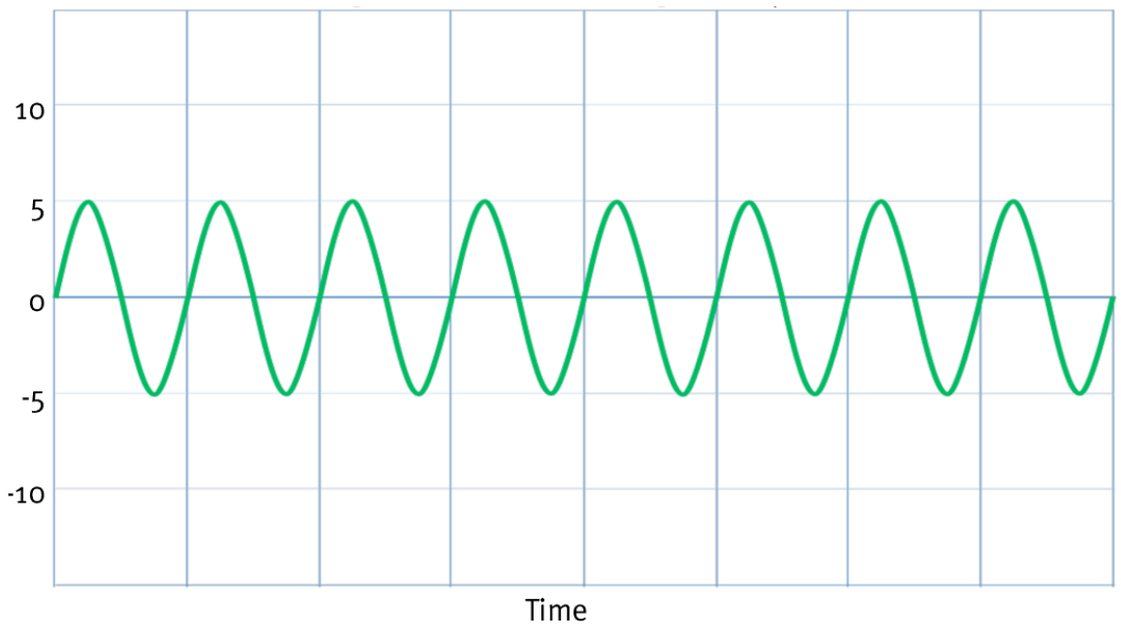
- Les leds ont deux pattes , la plus courte doit être placée sur le ground, la plus longue va vers la pin arduino.
- Vous avez besoin d'une résistance d'environ 10K ou 10000 ohms.
- Utilisez un breadbord pour pouvoir placer les éléments.

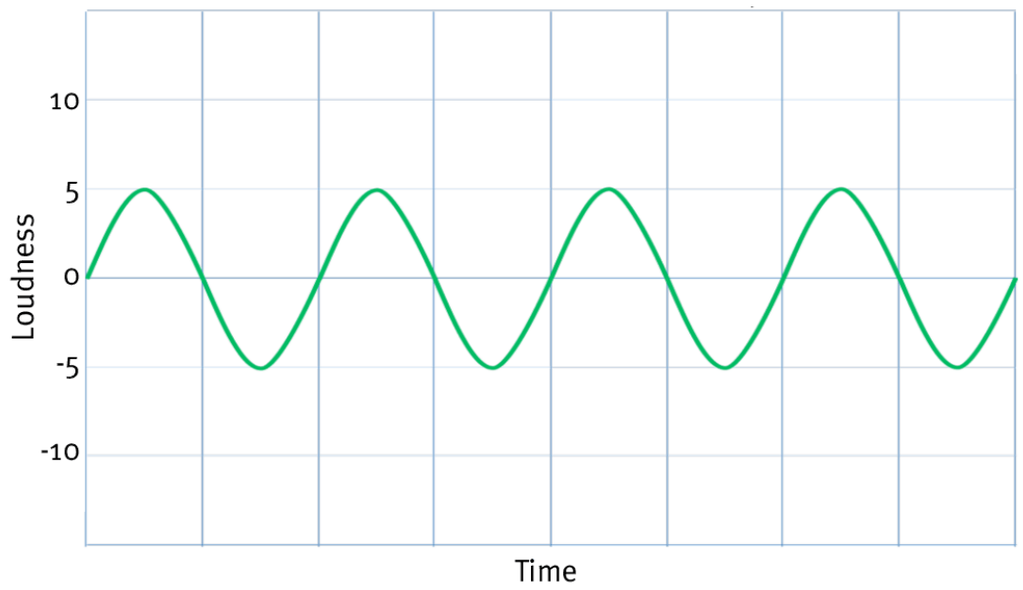
Output : son

Un autre output simple consiste a produire un son.

Un son n’est pas un simple passage de LOW a HIGH mais une fréquence, c’est à dire un passage rapide de LOW à HIGH.

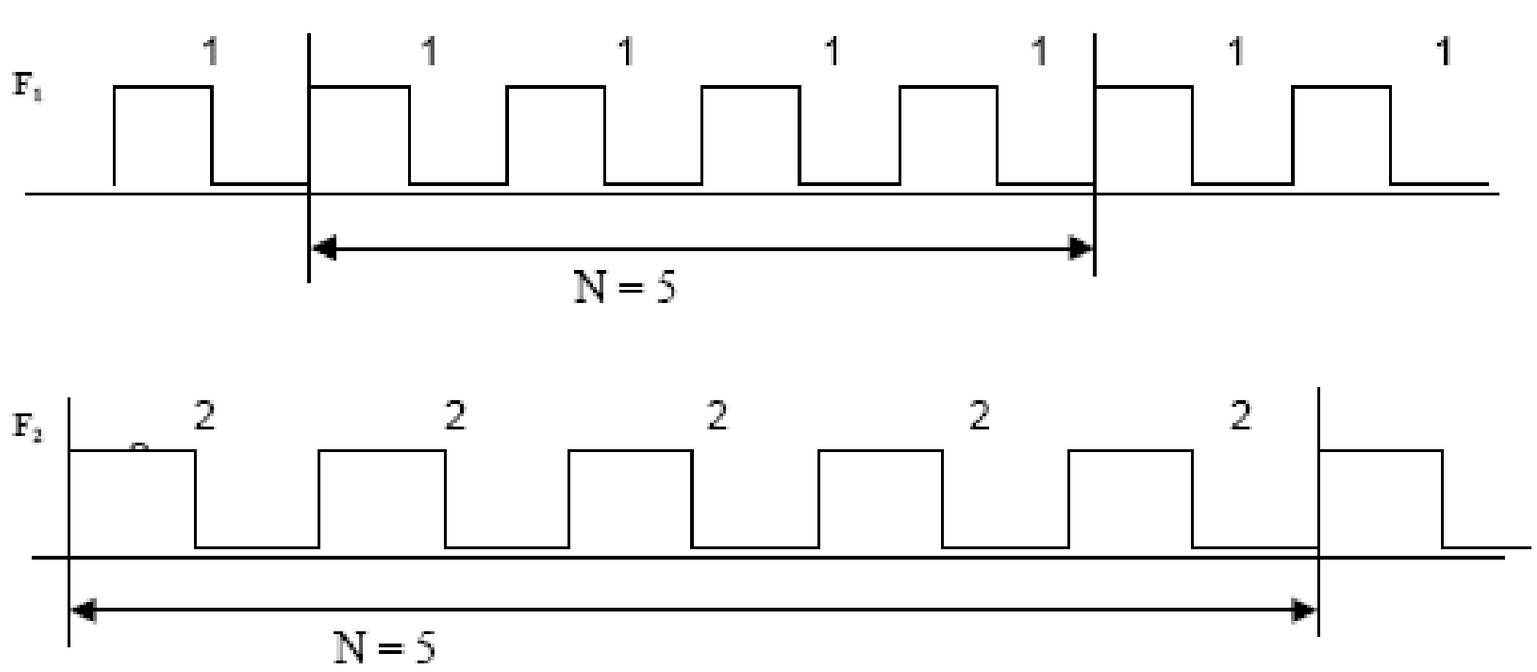
Voici deux sons analogiques.





Son numérique

Un son digital produit par Arduino est plus basique : c'est un passage de HIGH à LOW avec un temps de pose plus ou moins long pour obtenir des poses différentes.



Son numérique

Chaque note correspond à une pose entre HIGH et LOW différente. Voici une liste de notes et de fréquences :

B0 31	D2 73	F3 175	GS4 415	B5 988	D7 2349
C1 33	DS2 78	FS3 185	A4 440	C6 1047	DS7 2489
CS1 35	E2 82	G3 196	AS4 466	CS6 1109	E7 2637
D1 37	F2 87	GS3 208	B4 494	D6 1175	F7 2794
DS1 39	FS2 93	A3 220	C5 523	DS6 1245	FS7 2960
E1 41	G2 98	AS3 233	CS5 554	E6 1319	G7 3136
F1 44	GS2 104	B3 247	D5 587	F6 1397	GS7 3322
FS1 46	A2 110	C4 262	DS5 622	FS6 1480	A7 3520
G1 49	AS2 117	CS4 277	E5 659	G6 1568	AS7 3729
GS1 52	B2 123	D4 294	F5 698	GS6 1661	B7 3951
A1 55	C3 131	DS4 311	FS5 740	A6 1760	C8 4186
AS1 58	CS3 139	E4 330	G5 784	AS6 1865	CS8 4435
B1 62	D3 147	F4 349	GS5 831	B6 1976	D8 4699
C2 65	DS3 156	FS4 370	A5 880	C7 2093	DS8 4978
CS2 69	E3 165	G4 392	AS5 932	CS7 2217	

Codons un son continu

Les fréquences pour produire un son sont très rapides et en grand nombre. On utilisera la fonctiondelayMicroseconds (attention à la majuscule, elle est importante) :

```
void setup(){
    pinMode(8,OUTPUT);
}

void loop(){
    digitalWrite(8,HIGH);
    delayMicroseconds(440);
    digitalWrite(8,LOW);
    delayMicroseconds(440);
}
```

Brancher un haut-parleur sur la pin 8 et le ground.

Créer une fonction

Problème : le son est continu, et il s'agit d'une seule note. On va faire quelque chose de plus complexe : une fonction. On ajoute ceci sous le code précédent :

```
void note(int frequence){
      for(int i=0;i<200,i++){
            digitalWrite(8,HIGH);
            delayMicroseconds(frequence);
            digitalWrite(8,LOW);
            delayMicroseconds(frequence);
      }
}
```

Cette fonction est faite pour recevoir un chiffre, stocké dans une variable «frequence», et ensuite utilisé dans la fonction.

Nous avons créé notre propre fonction, «note()». Nous allons modifier le «loop()»

Utiliser la fonction tone()

Problème : le son est continu, et il s'agit d'une seule note. On va faire quelque chose de plus complexe : une fonction.

```
void setup(){
    pinMode(8,OUTPUT);
}

void loop(){
    note(440);
    delay(1000);
    note(220);
    delay(1000);
}

// ici plus bas la fonction note.
```

Une fonction toute faite

En fait une fonction existe depuis la version 1 de Arduino : la fonction tone():

tone(pin, frequency)
tone(pin, frequency, duration)

```
void setup(){
    pinMode(8,OUTPUT);
}

void loop(){
    tone(8,392,1000);
    delay(1000);
}

// ici plus bas la fonction note.
```

PROGRAMMATION

Les instructions

Une instruction est un ordre donné à la machine. Il faut respecter une syntaxe liée à l'environnement de programmation. Toute erreur de syntaxe provoque la plupart du temps l'arrêt de l'exécution du programme. Certaines erreurs courantes sont détectées par l'environnement de programmation de Arduino.

Une règle de base est de séparer les instructions par un ; appelé «semicolon» en anglais. On ne donne qu'une instruction à la fois, on la sépare d'une autre instruction par un semicolon.

Exemple :

```
int x=10;
```

Bases de la programmation

On vient d'utiliser rapidement 3 bases de la programmation : une fonction, une variable, une boucle.

Nous allons revenir sur ces bases, que l'on retrouve dans la quasi totalité des langages de programmation.

Les variables

Une variable est un espace de mémoire que l'on crée pour y stocker de l'information. Un chiffre, par exemple. On l'écrit dans l'espace de mémoire, et on peut ensuite le lire, et même le modifier.
Dans la plupart des langages, arduino y compris, on doit déclarer une variable avant de l'utiliser, ce qui signifie qu'on crée cet espace et que l'on indique de quel type il d'agira.

```
int y;
```

Signifie que l'on crée une variable du nom de y, qui contiendra des chiffres entiers.
On peut ensuite assigner une valeur :

```
y=10;
```

On peut faire les deux en une seule opération :

```
int y=10;
```

Les boucles

Une boucle permet de répéter une opération un certain nombre de fois. La boucle la plus courante est la boucle «for». Elle s'écrit comme suit

```
for(int i=0;i<250;i++){
// ici on répète 250 fois ce qui se trouve entre accolades
}
```

à l'intérieur des parenthèses,
- on crée une variable (i) que l'on initialise à 0.
- on donne une condition pour la répétition. Ici, tant que i est inférieur à 250.
- on dit ce qui se passe à chaque fois que la boucle se répète, ici que i augmente de 1

La boucle se répète tant que la condition n'est pas atteinte, et à chaque fois la valeur de i est modifiée.

Les conditions

Les conditions permettent d'exécuter du code uniquement si certaines conditions sont remplies.

On les écrit comme ceci :

```
if(condition) {
// code a exécuter
} else {
// code a executer si la condition n'est PAS remplie
}
```

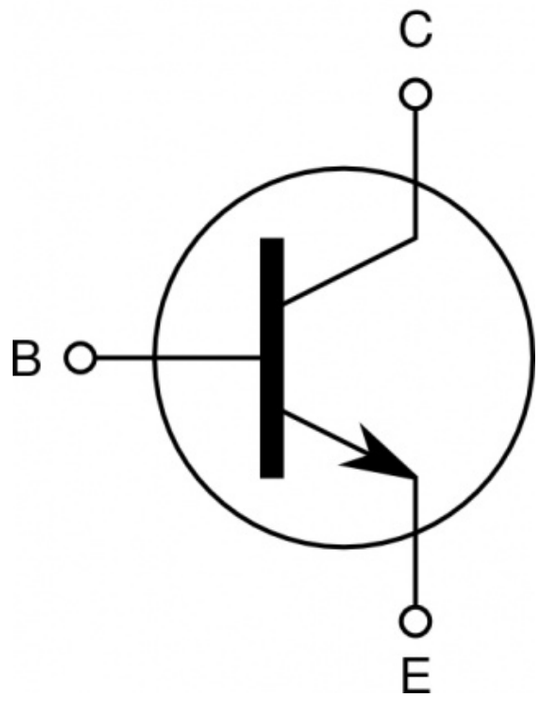
exemple

```
if(a ==0){
digitalWrite(13,HIGH);
}
```

Output : plus de puissance

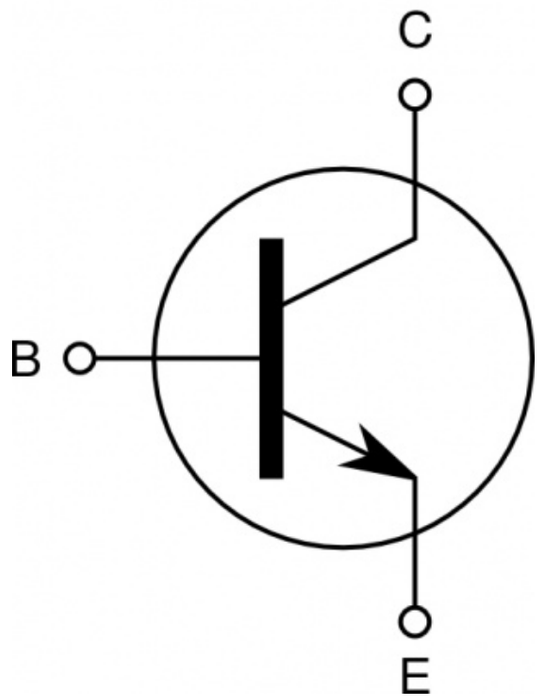
Arduino ne peut pas piloter directement des actuateurs demandant un fort ampérage. Il existe des solutions cependant, celle de passer par un transistor ou encore par un relai.

Il faut alors faire un montage.



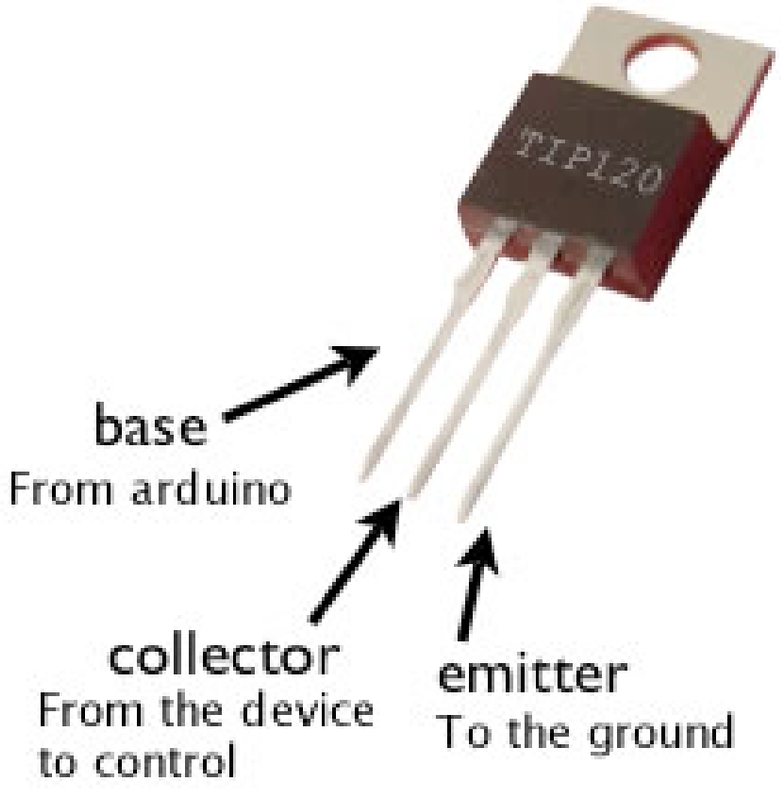
Un transistor

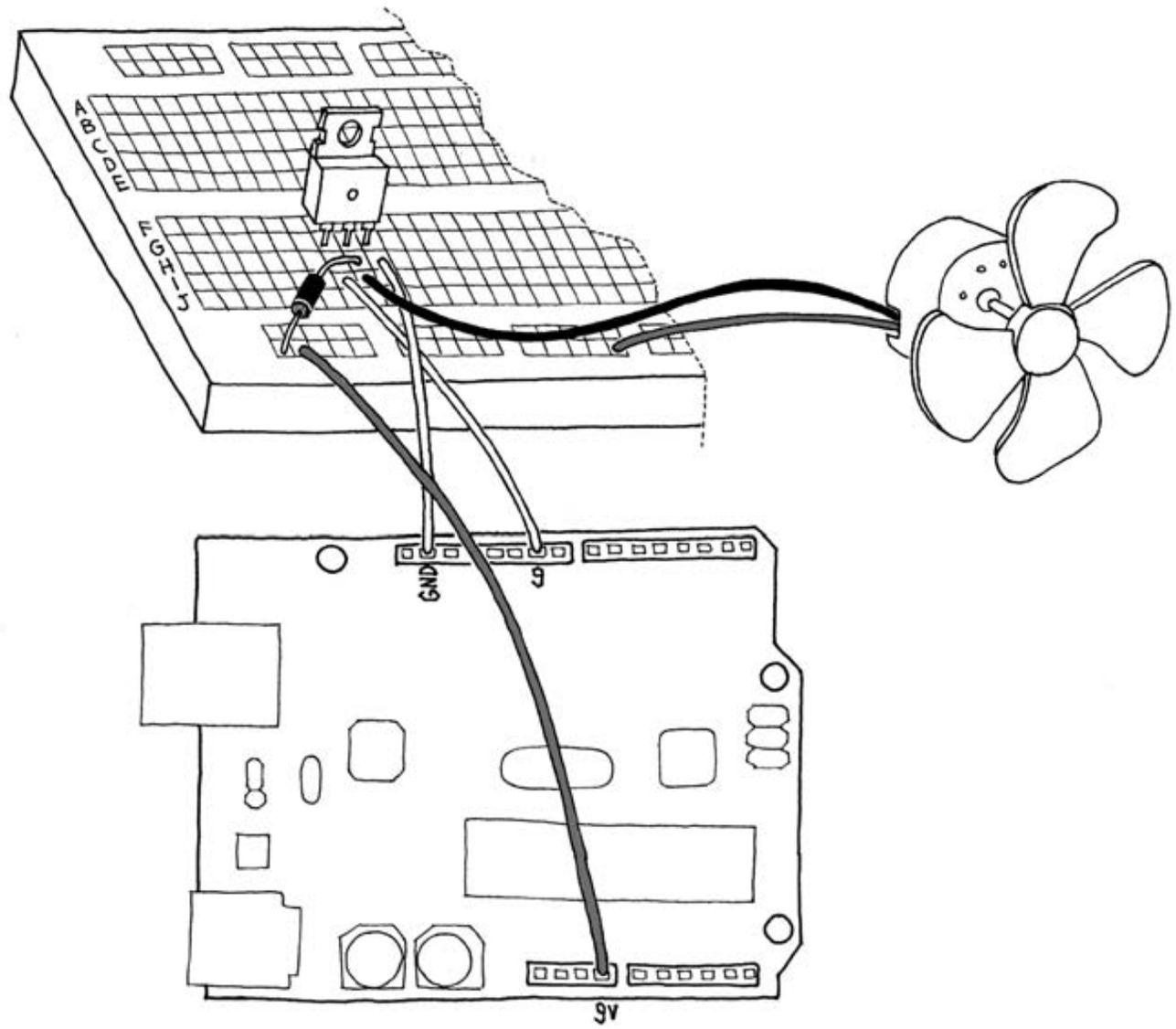
Un transistor permet de piloter plus de puissance. C’est une sorte de vanne électronique. Arduino tourne la vanne, et un courant plus puissant peut passer. C’est par ce biais qu’un simple circuit permet de contrôler une centrale électrique.



Output : plus de puissance

Voici un montage utilisant un mosfet. Il permet de piloter des moteurs et autres petits actuateurs jusqu'à 12 volts plus ou moins,
des lectro-aimants par exemple.



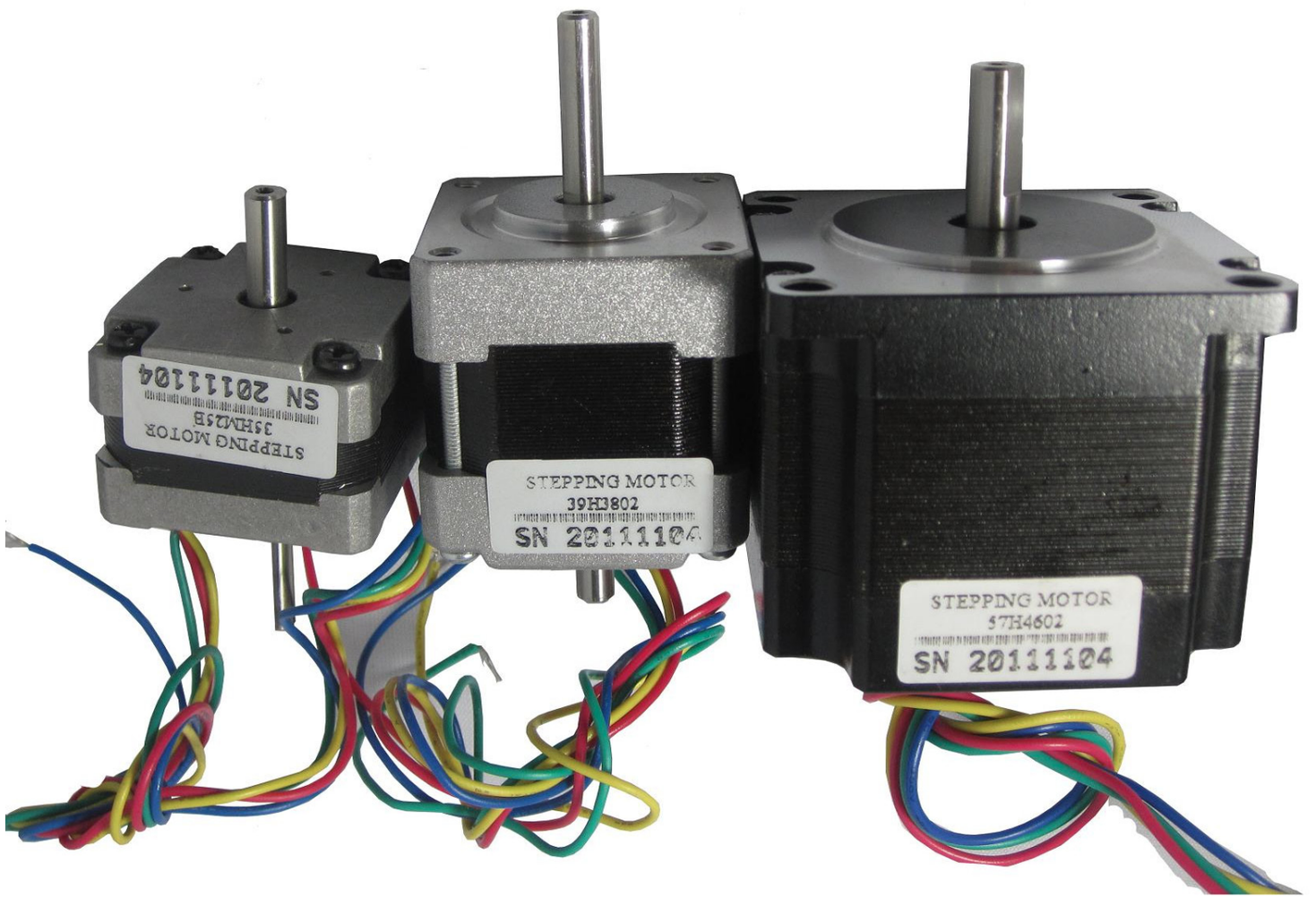


Output : DC motor

Un moteur DC peut être trouvé dans beaucoup de petits jouets. Ils tournent vitent, sont bruyants, consomment beaucoup et sont difficiles à contrôler. Mais il y a tout de même des choses à faire avec.

Output : stepper motor

Un stepper est un moteur plus complexe nécessitant un montage plus complet. En contrepartie, il peut être précis. En fait, il peut avancer par pas, en avant ou arrière. On en trouve dans la plupart des imprimantes.



Output : servo motor

Un servo motor est un moteur plus complexe, comportant généralement des rouages et un système de pilotage qui le rend précis. Il est très facile de les utiliser avec arduino.