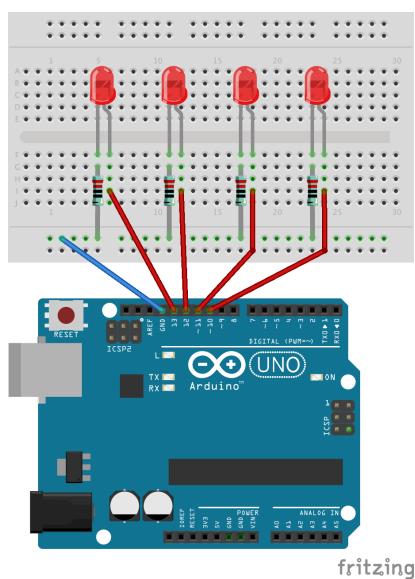


# Arduino

*Transmettre aux  
enfants le goût de la  
programmation, de la  
robotique et de  
l'ingénierie*

## Atelier Arduino Lumière



Si cet atelier vous a plu, vous pouvez refaire cet atelier chez vous avec un budget de moins de 100 €.

Il requiert :

- un Arduino Uno ou un clone (environ 15 €)
- une planche d'essai (breadboard)
- quelques LEDs
- un buzzer
- quelques résistances 220 Ω
- des fils
- pour les ateliers plus avancés une photo-résistance et une résistance 330 Ω

Il existe un starter kit officiel Arduino mais vous pouvez trouver ces composants dans presque tous les starter kits (nous avons des starter kits Farnell et Funduino).

Les supports sont disponibles sur le dépôt de documents de Devoxx4kids France

<http://devoxx4kidsfr.github.io/materials/ateliers/arduino/index.html>

Vous trouverez des informations plus générales sur nos activités et les différents ateliers sur le site Web de Devoxx4Kids France

<http://www.devoxx4kids.org/france/ateliers/arduino/>

L'alphabet morse ou code morse, est un code permettant de transmettre un texte à l'aide de séries d'impulsions courtes et longues, qu'elles soient produites par des signes, une lumière ou un son.

Tu es perdu et ton objectif est de construire un appareil qui émet le signal de détresse S.O.S. toutes les 20 secondes.

*Nous aurons besoin de*



*L'Arduino Uno*



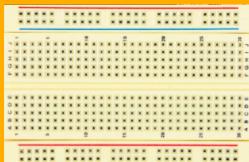
*1 LED*



*Le cable USB*



*L'ordinateur*



*La planche d'essai*



*1 résistance  
220 Ω*



*Le programme  
Blink*

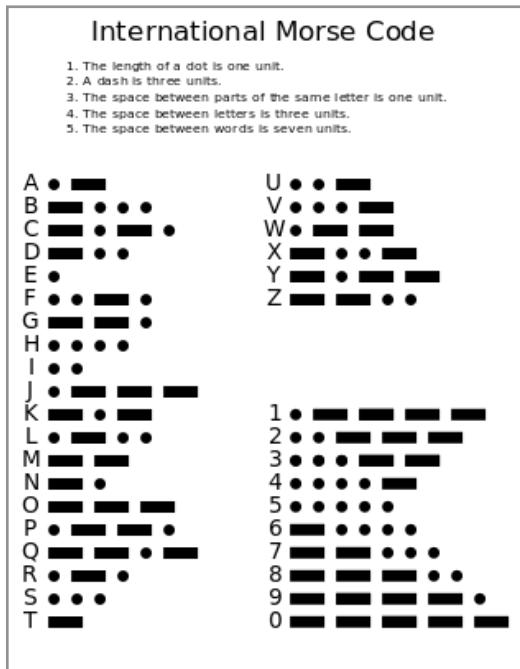


*L'éditeur de programme*

Le **programme Blink** est un des programmes exemple. Il fait clignoter une LED connectée au pin 13.

Tu trouveras le montage sur la fiche “Premier Circuit” et des explications supplémentaires sur le programme dans les fiches “Le programme Blink”, et les fonctions.

L'alphabet Morse est fourni au dos de cette fiche.



L'alphabet morse utilise des impulsions courtes et longues.

Par exemple pour faire S tu dois faire 3 courts, c'est à dire faire 3 fois la suite d'opérations allumer la LED pendant 1s - attendre 1s - éteindre la LED.

Pour un long tu devra allumer la LED pendant 3s.

Réouvre le programme Blink et sauve le sous le nom Morse avec le bouton le plus à droite et qui s'appelle "Save".

Pour que ça soit plus facile on écrira des fonctions pour chaque type de signal et chaque lettre.

```
void allumeLED() {
    digitalWrite(13, HIGH); // utilise LOW pour éteindre
}

void attend(int tempsEnSecondes) {
    delay(tempsEnSecondes * 1000);
}

void signalCourt() {
    allumeLED();
    attend(1);
    eteintLED();
    attend(1);
}

void signalS() {
    signalCourt();
    signalCourt();
    signalCourt();
}
```

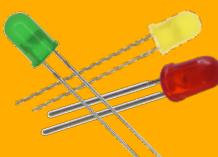
Avec plusieurs LEDs on peut faire une guirlande de Noël



*Nous aurons besoin de*



L'Arduino Uno



4 LEDs



4 résistances  
 $220\ \Omega$



Le programme  
GuirlandeDeNoel

Ouvre le programme GuirlandeDeNoel. Utilise le premier menu Ouvre... et cherche le fichier GuirlandeDeNoel.ino.

Le principe est que les LED s'allument à tour de rôle.

Les LEDs sont sur les pins 10 à 13. La seule chose que l'on va retenir c'est le décalage à partir du pin 10, c'est à dire 0, 1, 2 ou 3.

On va conserver cette valeur dans une **variable** `decalage`. On pourra recalculer le numéro de pin en ajoutant `PIN_BASE` qui est le numéro du premier pin utilisé. La valeur de `PIN_BASE` ne change pas, c'est une **constante**.

```
int PIN_BASE = 10;  
int decalage = 0;
```

Dans `setup`, on va déclarer les pins utilisés en utilisant une boucle `for`. Elle va générer toutes les valeurs à partir de 0 et tant que la valeur n'atteint pas `NOMBRE_DE_LEDS`.

```
int NOMBRE_DE_LEDS = 4;

void setup () {
    for (int i=0; i<NOMBRE_DE_LEDS; i++) {
        pinMode (PIN_BASE + i, OUTPUT);
    }
}
```

Dans `loop`, on va augmenter le décalage de 1 à chaque passage, mais en le gardant inférieur à 4. Pour ça on utilise la fonction `%` (modulo).. Elle calcule le reste de la division par 4. Fait quelques essais pour voir comment marche modulo.

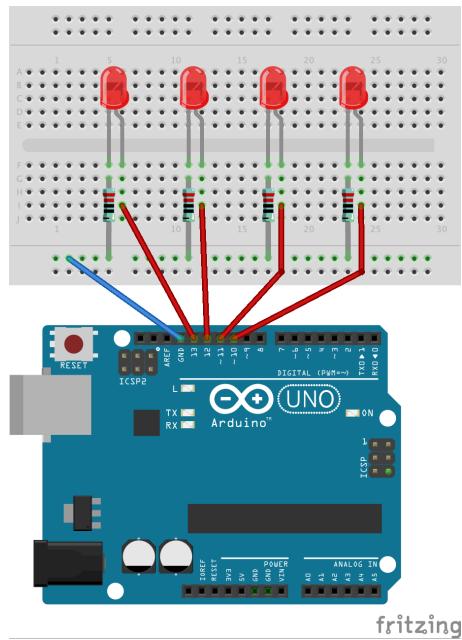
```
void loop() {
    digitalWrite(PIN_BASE + decalage, LOW); // eteint
    decalage = ((decalage + 1) % NOMBRE_DE_LEDS);
    digitalWrite(PIN_BASE + decalage, HIGH); // allume
    delay(100);
}
```

Pour voir ce qui se passe tu peux utiliser `Serial` et l'outil `Serial Monitor`

```
// dans setup
Serial.begin(9600);
// dans loop
Serial.println("Decalage");
Serial.println(decalage);
```

Tu peux changer la vitesse de la guirlande en modifiant le paramètre de `delay`.

Voici le circuit. Fait bien attention au sens des LEDs, aux résistances utilisées et aux pins (10 à 13).



Les premiers ordinateurs n'avaient pas d'écran. Ils affichaient les résultats des calculs en allumant des lampes.

C'est pour ça que les vieux films de Science-Fiction ont des ordinateurs très bizarres.



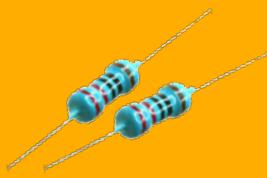
*Nous aurons besoin de*



L'Arduino  
Uno



4 LEDs



4 résistances  
220 Ω



La fiche bits  
et base 2

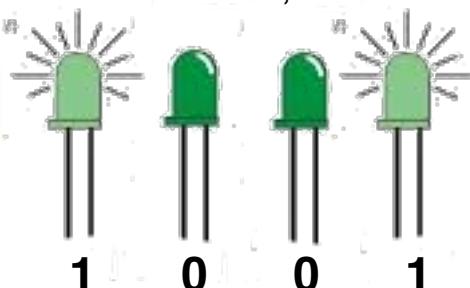


Le programme  
Univac

L'Arduino, tout comme les premiers ordinateurs ne connaissent que deux valeurs : 0 et 1. Le courant ne passe pas (LOW), ou il passe (HIGH).

Les nombres sont représentés en base 2. Tu trouveras toutes les explications dans la fiche "Bits et base 2".

On va placer 4 LEDs et afficher chaque bit du nombre sur une des lampes. L'Arduino fourni une fonction **bitRead** qui permet de lire le bit à une position donnée. On va l'appeler pour les positions 0 à 3, et on calculera la valeur à envoyer sur le pin



## Charge le programme univac.ino.

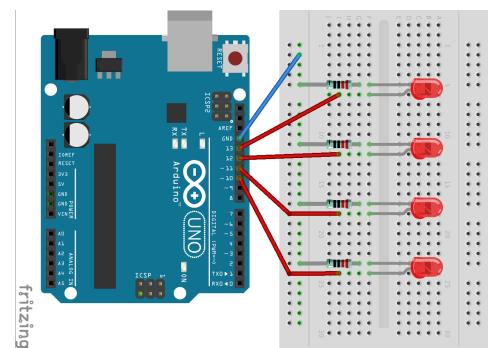
On va se servir du résultat de **bitRead** pour calculer la valeur de l'on applique sur le pin.

- pour bit==1, 1 x HIGH donne la même valeur que HIGH
- pour bit==0, 0 x HIGH vaut 0, donc la valeur que la constante LOW

```
for (int i=0; i<NOMBRE_DE_LEDS; i++) {  
    byte bit = bitRead(unNombre, i);  
    digitalWrite(PIN_BASE + i, bit * HIGH);  
}
```

On va faire une fonction **afficheUnNombre** qu'on pourra réutiliser.

Pour le circuit, c'est le même que celui de la guirlande



Dans un premier temps, change le nombre passé à **afficheUnNombre** et vérifie ce qui s'affiche. N'oublie pas de télé-verser le programme à chaque modification.

Le nombre le plus grand qu'on pourra afficher en base 2 avec 4 LEDs est **15**. Si tu as bien étudié la fiche "Bits et base 2" tu dois savoir pourquoi 15.

**Serial** te permettra d'afficher les bits que tu calcule et de rentrer de nouveaux nombres.

On va se servir de l'affichage de nombre en binaire de l'interface Old School pour allumer les LEDs de la guirlande et faire des motifs.



*Nous aurons besoin de*



L'Arduino Uno



4 LEDs



4 résistances  
220  $\Omega$



Le programme  
Univac

Reprend le circuit et le programme de la fiche Interface Old School. Sauve le programme sous un autre nom.

On va rajouter des LEDs de toute les couleurs. Pour cela il faut adapter le programme pour qu'il marche avec n'importe quel nombre de LEDs.

Cette valeur peut être changée facilement en modifiant la constante NOMBRE\_DE\_LEDS.

```
int NOMBRE_DE_LEDS = 4;
```

Si le nombre de LEDs change, il faudra adapter les pins utilisés et la valeur MAX.

La fonction afficheUnNombre utilise déjà la constante et elle n'a pas besoin d'être modifiée.

Pour 4 LEDs, on utilisait les pins 10, 11, 12 et 13.

Pour calculer le premier pin (10 pour 4 LEDs) de manière simple, tu peux utiliser la formule suivante :

```
int PIN_BASE = 14 - NOMBRE_DE_LEDS;
```

Pour obtenir un nombre au hasard, l'Arduino fournit une fonction `random` qui donne une valeur entre 0 et le nombre précédent le paramètre. Par exemple, la ligne suivante donne un nombre au hasard entre 0 et 15. On va aussi déplacer l'affichage dans `loop`.

```
void loop() {  
    int nombre = random(LIMITE);  
    afficheUnNombre(nombre);  
    delay(100);  
}
```

Comme le nombre de LEDs peut changer, il faut calculer la limite au début du programme.

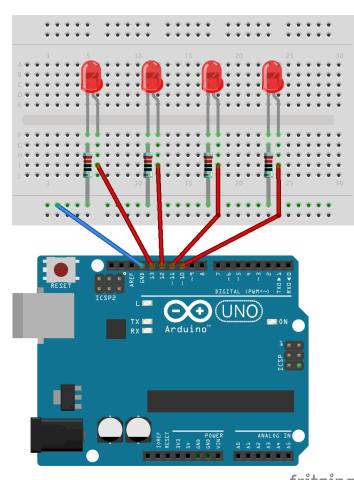
```
int LIMITE = 2 << NOMBRE_DE_LEDS;
```

Si on a 4 LEDs on pourra afficher 4 chiffres. Il y a 16 nombres de 4 chiffres en base 2. Ce nombre peut être calculé par  $2^2 \cdot 2^2 \cdot 2^2$ , qui se dit aussi 2 puissance 4. Si tu recompte ça fait bien 16.

**L'opérateur** `<<` correspond à faire `*2`. L'opération `2 << 2` donne 4, `2 << 3` donne 8, et `2 << 4` donne 16.

Tu peux utiliser `Serial` pour afficher les valeurs.

Le circuit est le même que pour la Guirlande de Noël.



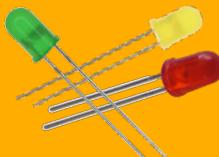


On va se servir de l'affichage de nombre en binaire de l'interface Old School pour allumer les LEDs à la façon d'une enseigne lumineuse.

*Nous aurons besoin de*



L'Arduino Uno



4 LEDs



4 résistances  
 $220\ \Omega$



Le programme  
GuirlandeFolle

Reprend le circuit et le programme de la fiche Guirlande Folle. Sauve le programme sous un autre nom.

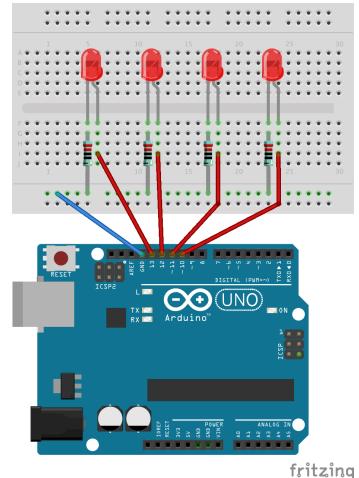
On va rajouter un tableau qui décrit la séquence à jouer. On va utiliser les nombres en binaire pour représenter les formes à afficher.

Par exemple, avec 4 LEDs, pour allumer les 2 LEDs à gauche (💡💡🔴🔴), il faut le nombre binaire 1100 ce qui fait  $8+4=12$ .

Une séquence



s'écrit 12, 0, 3, 0



Le circuit est le même que pour la Guirlande de Noël.

Ecrit ta propre séquence et place la dans la constante SEQUENCE.

Une variable ou une constante qui est suivie de [] est un **tableau**. Les signes [] s'appellent des **crochets**. Dans le cas le plus simple, on va indiquer les valeurs à la suite de la déclaration de la variable entre { }. Les signes { } s'appellent des **accolades**)

```
byte SEQUENCE[] = { 3, 0, 12, 0 };
```

Pour **accéder à une valeur du tableau**, on utiliser aussi [] et la position. Attention, la position de la première valeur est 0.

```
afficheUnNombre(SEQUENCE[pos]);
```

Après chaque lecture il faudra avancer la position à laquelle on lit. La fonction % (modulo) permet de s'assurer que le nombre reste dans les limites.

Il faudra calculer le **nombre d'éléments** du tableau dans le setup. Il n'y a pas de fonction qui fait ça dans ce langage. Il faut diviser la place que prend le tableau (sizeof) par la place que prend chaque élément, ici des bytes.

```
byte l = sizeof(SEQUENCE) / sizeof(byte)
```

Les Push Buttons permettent de fermer ou ouvrir le circuit sur commande.



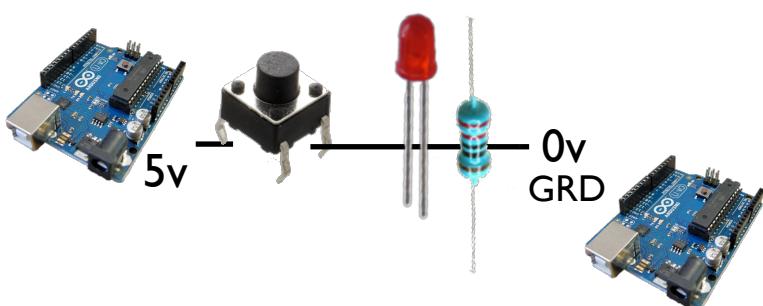
On l'utilise comme interrupteur marche/arrêt ou pour déclencher des opérations.

### Comment ça fonctionne ?

L'Arduino fourni une tension de 5V (volt) en entrée. Lorsque le bouton n'est pas utilisé, ses pattes ne sont pas reliées et le circuit est ouvert. Le courant ne passe pas.

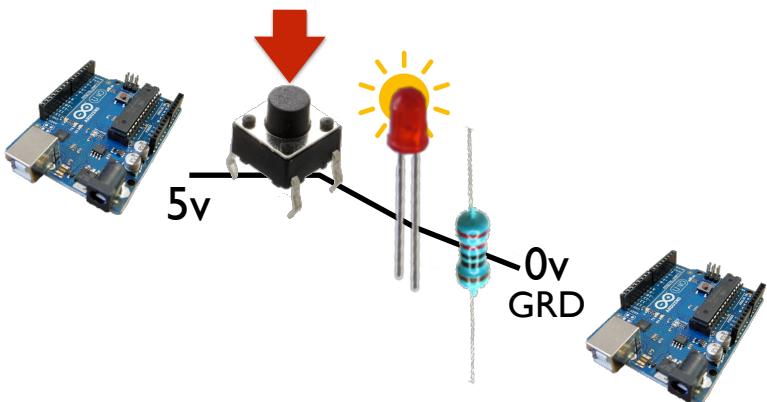
Lorsque l'interrupteur est actionné, les pattes sont reliées et le circuit est fermé. Le courant passe.

#### Cas 1 : bouton est au repos



Le push button ne laisse pas passer le courant.

#### Cas 2 : le bouton est enfoncé



Le push button laisse passer le courant.

## Comment lire le résultat ?

Si on lit la tension après le push button, la valeur sera HIGH si le bouton est pressé ou LOW si le bouton n'est pas utilisé et le circuit est ouvert. Pour cela, on peut utiliser *digitalRead*.



Il faudra lire la valeur régulièrement dans loop par

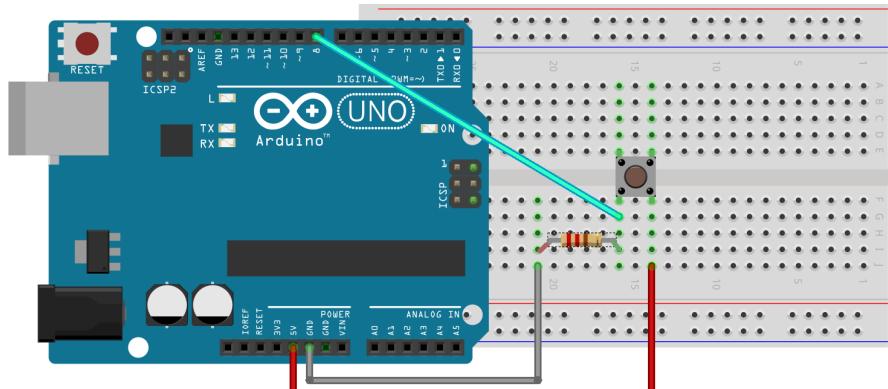
```
int valeur = digitalRead(8);  
if (valeur == HIGH) { ..... }
```

Il peut être difficile d'isoler chaque clic si l'utilisateur tremble un peu. Il faut ajouter un stabilisateur :

```
int valeur = digitalRead(8);  
if (valeur == HIGH) {  
    unsigned long temps = millis();  
    if (temps - tempsPrecedent > 1000) {  
        tempsPrecedent = temps;  
        ...  
    }  
}
```

## Le circuit

On ajoute une résistance en sortie pour consommer le courant restant.



fritzing