Avec les Bons tout devient facile !

2e édition

Arduino pour les bons

- Trouver une carte et l'installer
- S'initier à l'électronique et travailler avec les croquis
- Apprentissage de la soudure
- Cartes filles et bibliothèques
- Gestion optimale des entrées/sortie

Hisoka Morow

Bienvenue dans le guide. Ce guide se veut informatif et facile à comprendre. N'hésitez pas à ajouter des commentaires pour que ce guide puisse être amélioré. Ce guide est fortement inspiré du site d'Arduino https://docs.arduino.cc/built-in-examples/. N'oubliez pas, si jamais vous voulez coder quelque chose, il y a sûrement quelqu'un sur Internet qui l'a déjà fait.

Contents

Minimum pour un code Arduino

Un code pour Arduino doit contenir la fonction setup et la fonction loop au minimum.

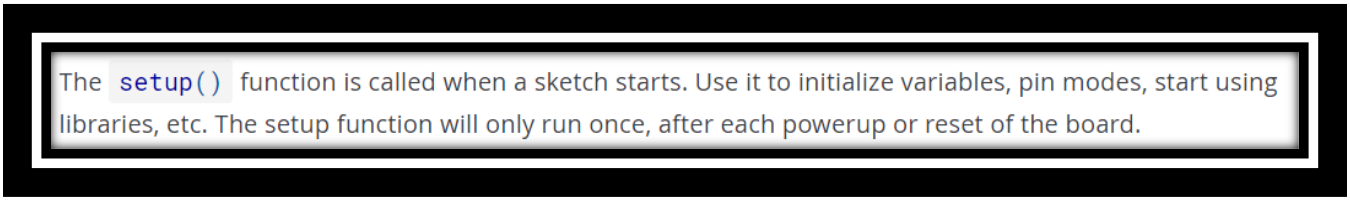
Bare Minimum code needed

The bare minimum of code needed to start an Arduino sketch.

Last revision • 02/24/2024

This example contains the bare minimum of code you need for a sketch to compile properly on Arduino Software (IDE): the `setup()` method and the `loop()` method.

La fonction setup est appelée une fois au début du programme. Elle sert à initialiser des variables, des types de pins (entrée, sortie), le port série, etc.



The `setup()` function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the board.

La fonction loop, elle, comme son nom l'indique, tourne en boucle pendant l'exécution du programme. C'est un peu l'équivalent du main. Ce qui suit deux barres obliques (//) est considéré comme un commentaire et ne va pas être pris en compte par le compilateur. Voici une bonne structure de départ pour commencer à coder.

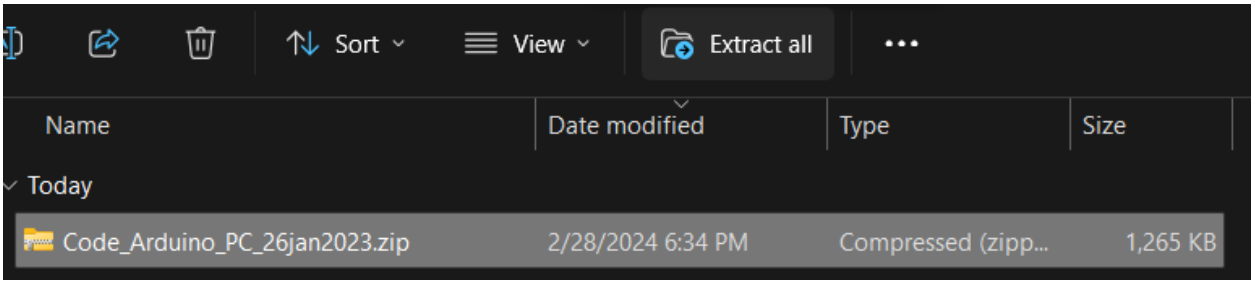
```
1  void setup() {
2    // put your setup code here, to run once:
3  }
4
5  void loop() {
6    // put your main code here, to run repeatedly:
7  }
8
```

Ouverture du code sur VS Code

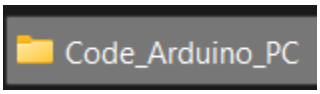
VS Code possède plusieurs extensions qui peuvent faciliter la vie. Par exemple, au lieu de coder dans l'IDE d'Arduino, vous pouvez directement coder dans VS Code avec l'extension PlatformIO. Normalement, vous devriez l'avoir déjà installée. Si vous avez des problèmes pour l'installer, c'est mieux de régler ces problèmes en présentiel.

Dans le cadre du projet de session 2, les profs fournissent déjà un gabarit de départ pour le code dans le site de session dans la section du projet. Ils fournissent même un guide pour l'installation du compilateur MinGW et comment compiler le code et l'exécuter.

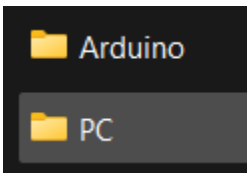
Après avoir téléchargé le dossier zip, il faut l'extraire.



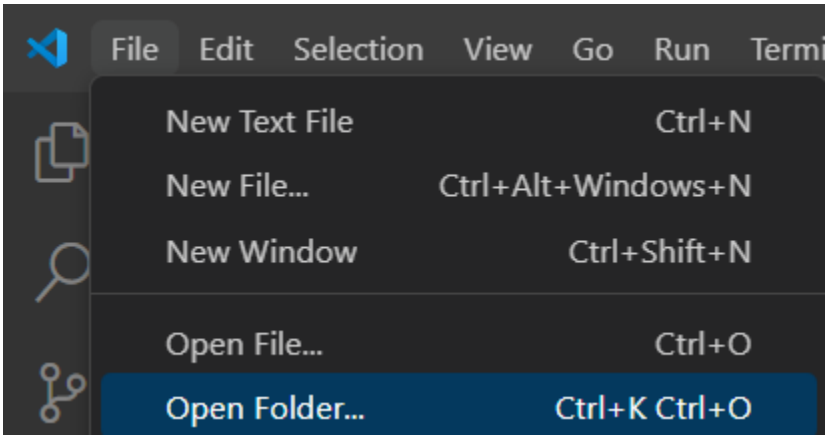
Vous vous retrouverez alors avec un dossier Code_Arduino_PC (à l'intérieur d'autres dossiers).



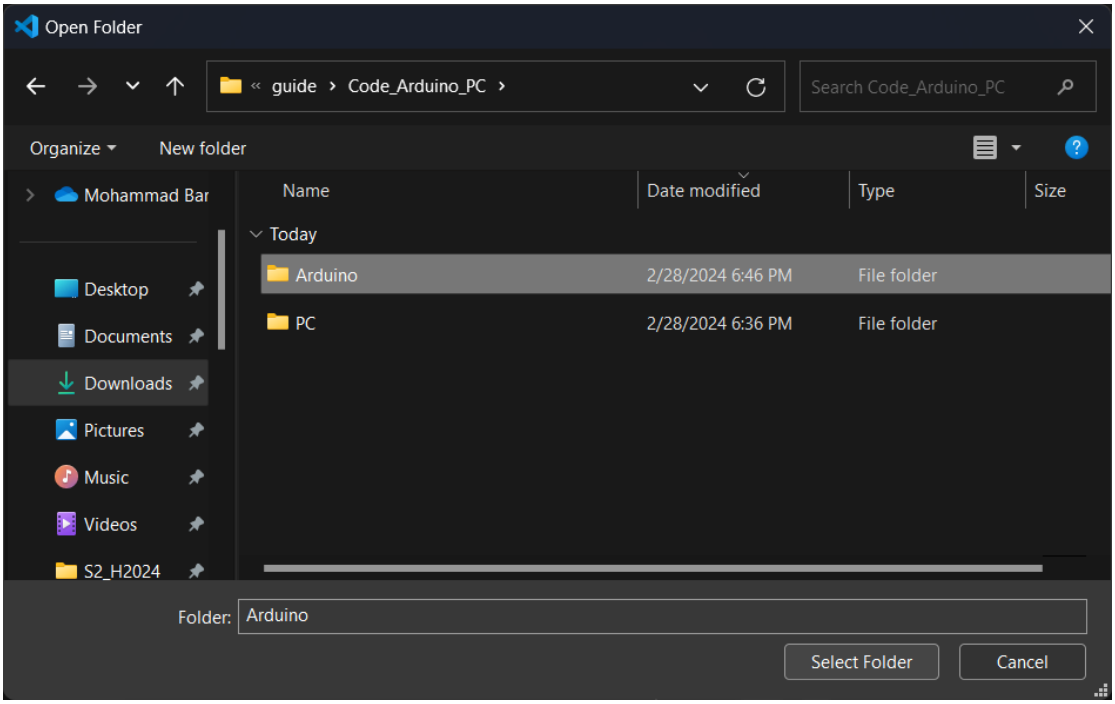
Dans ce dossier, il y a un dossier pour le code Arduino et un autre pour le code du jeu sur PC.



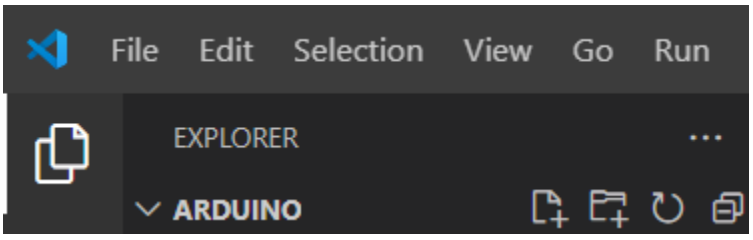
Il suffit d'ouvrir VS Code. Aller dans File > Open Folder



Sélectionner le dossier Arduino.

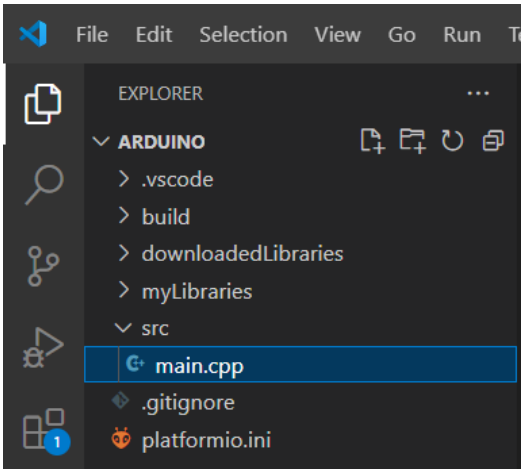


Vous vous retrouvez avec le dossier ouvert dans VS Code.



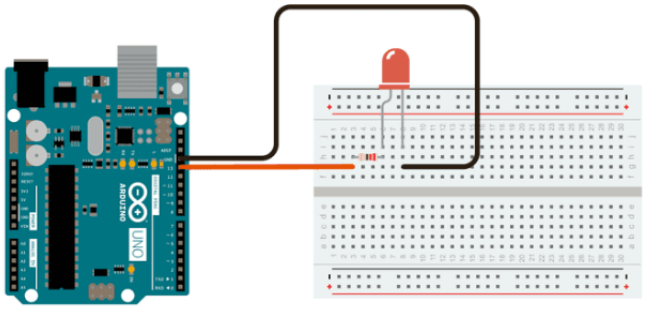
Codes Arduino de base sur VS Code

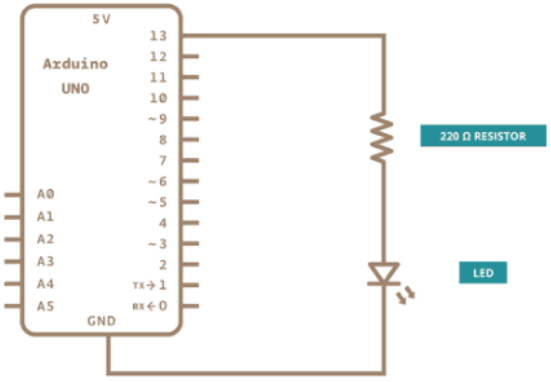
Le code se trouve dans le dossier src et se nomme main.cpp.



Faire clignoter une DEL

Pour faire clignoter une DEL, il faut d'abord brancher une DEL à l'Arduino. Il faut mettre une résistance (1k par exemple) pour ne pas griller la DEL. Il faut utiliser une des nombreuses pins numériques « Digital » pour envoyer le signal à la DEL. N'oubliez pas de fermer le circuit en connectant au Ground. Une pin numérique fonctionne à 0 ou 5V, contrairement à une pin analogique qui marche de 0 à 5V. En général, la patte la plus longue d'une DEL est la patte +, celle que vous connectez au signal de l'Arduino.





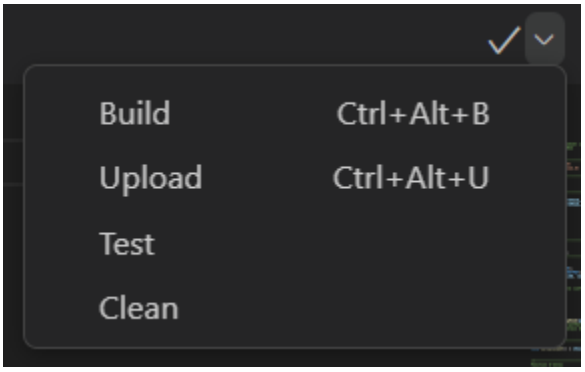
Pour ce qui est du code, il faut d'abord déclarer la patte pour la DEL en sortie. Dans la fonction setup, on utilise donc la fonction pinMode. On spécifie d'abord le numéro de la pin, par exemple 22, puis le type (entrée ou sortie). Ici, on déclare la DEL déjà sur l'Arduino en OUTPUT (LED_BUILTIN est un define).

Dans la loop, on allume la DEL pendant une seconde, puis on l'éteint pendant une seconde. Cela donne un clignotement. Pour allumer la DEL, on utilise la fonction digitalWrite. Comme pour pinMode, on spécifie d'abord le numéro de la pin. Ensuite, on spécifie l'état (HIGH ou LOW). C'est la même chose pour l'éteindre. La fonction delay, quant à elle, permet de bloquer le programme. Cela crée une attente. Voici le code.

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

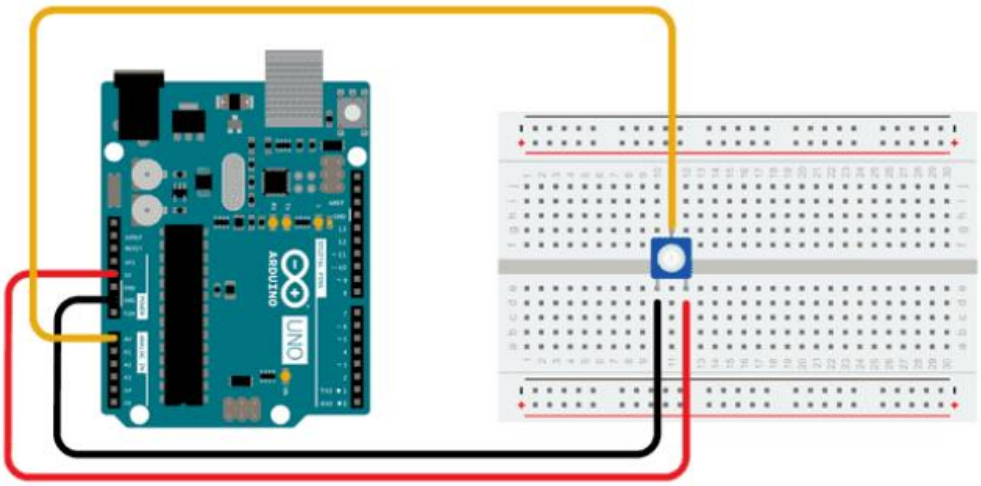
Téléverser le code

Dans VS Code, vous pouvez compiler le code, puis le téléverser. Petit truc, Upload compile déjà le code, donc, pas besoin de build avant de upload. Voici une photo du coin en haut à droite dans VS Code.



Lire un potentiomètre

Pour lire un potentiomètre, il faut d'abord le brancher. Connecter une extrémité au 5V et l'autre au GND. Connectez la patte du milieu à une pin analogique (par exemple A0) du Arduino. Pourquoi une pin analogique? Si on la connectait à une pin digital, on aurait LOW ou HIGH comme réponse. Ce n'est pas ça qu'on veut. On veut plus de valeurs possibles pour le potentiomètre.



Le code qui suit permet de lire un potentiomètre et d'afficher le résultat (0-1023) dans le moniteur série.

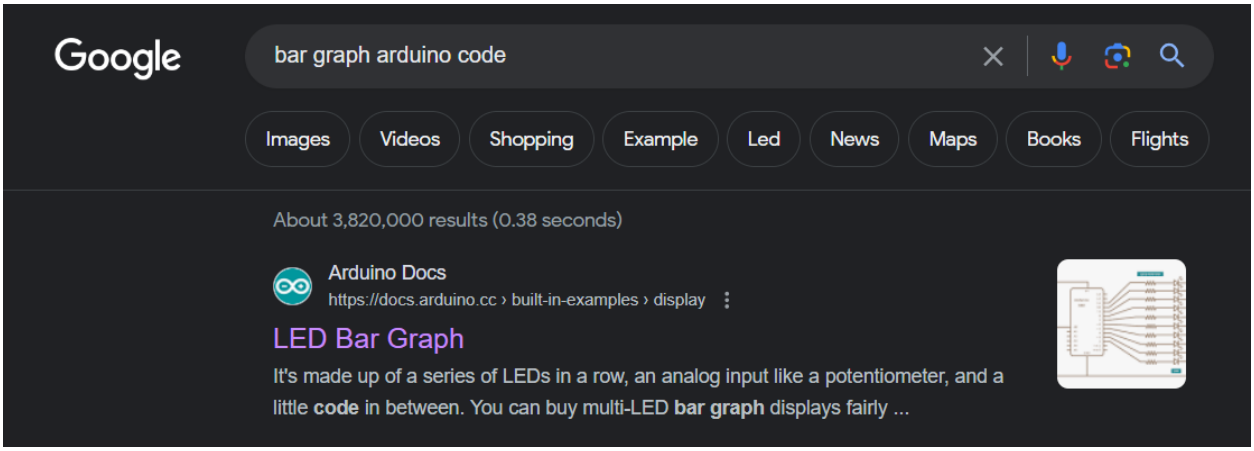
```
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);  // delay in between reads for stability
}
```

Dans le setup, il faut initialiser la communication série avec Serial.begin à la vitesse de 9600 bauds.

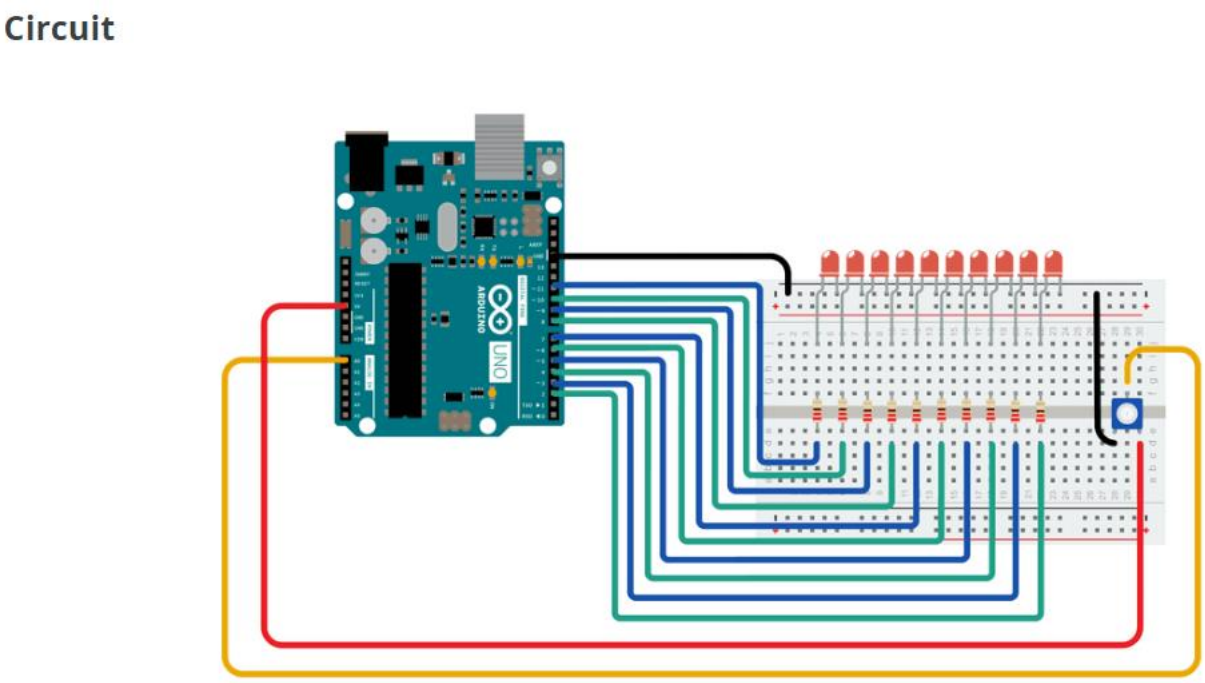
Dans la loop, on peut créer une variable de type int qui va stocker la valeur lue du pot. Serial.println affiche la valeur entre parenthèses dans le moniteur série. Le ln à la fin du Serial.print signifie « new line ». Il y aura donc un saut de ligne à chaque affichage. Finalement, il y a un délai de 1 seconde entre les lectures. Normalement, il faudrait aussi déclarer la patte A0 du pot en INPUT dans le setup, mais les pattes analogiques sont automatiquement en entrée. En effet, elles ne vont pas te sortir la tension que tu veux entre 0 et 5V. Elles ne peuvent que lire une valeur. Attention, vous ne verrez pas une valeur entre 0 et 5 dans le moniteur, mais bien entre 0 et 1023. C'est ce que le convertisseur sort comme valeur. Il est possible de faire un petit produit croisé pour convertir après en tension.

Chercher du code sur Internet

Ma tâche, c'était de m'occuper du Bar Graph et du potentiomètre. Voici une manière simple de s'y prendre. J'ai cherché « bar graph arduino code » sur Internet et j'ai cliqué sur le premier lien, le site officiel d'Arduino (les autres sites aussi sont bons).



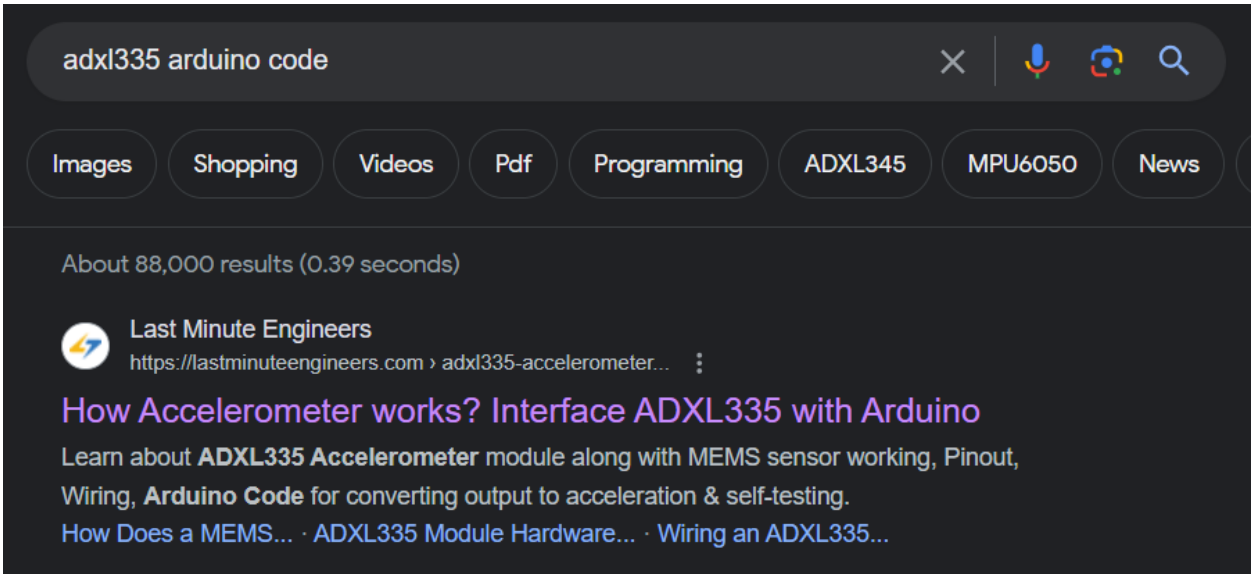
Dans la page, ça m'expliquait comment brancher le tout.



Ça me donnait même un code pour tester. Ce code permet d'allumer de plus en plus de DELs sur le Bar Graph selon la valeur du potentiomètre. En tournant le pot, les DELs s'allument.

```
// these constants won't change:
const int analogPin = A0;  // the pin that the potentiometer is attached to
const int ledCount = 10;   // the number of LEDs in the bar graph

int ledPins[] = {
  2, 3, 4, 5, 6, 7, 8, 9, 10, 11
};  // an array of pin numbers to which LEDs are attached


void setup() {
  // loop over the pin array and set them all to output:
  for (int thisLed = 0; thisLed < ledCount; thisLed++) {
    pinMode(ledPins[thisLed], OUTPUT);
  }
}

void loop() {
  // read the potentiometer:
  int sensorReading = analogRead(analogPin);
  // map the result to a range from 0 to the number of LEDs:
  int ledLevel = map(sensorReading, 0, 1023, 0, ledCount);

  // loop over the LED array:
  for (int thisLed = 0; thisLed < ledCount; thisLed++) {
    // if the array element's index is less than ledLevel,
    // turn the pin for this element on:
    if (thisLed < ledLevel) {
      digitalWrite(ledPins[thisLed], HIGH);
    }
    // turn off all pins higher than the ledLevel:
    else {
      digitalWrite(ledPins[thisLed], LOW);
    }
  }
}
```

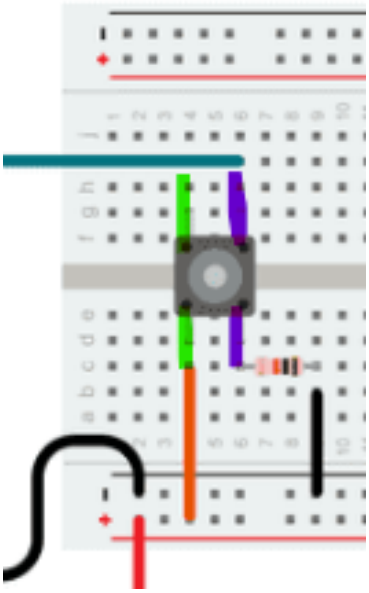
On peut faire la même chose pour l'accéléromètre, etc.



Lire un bouton-poussoir

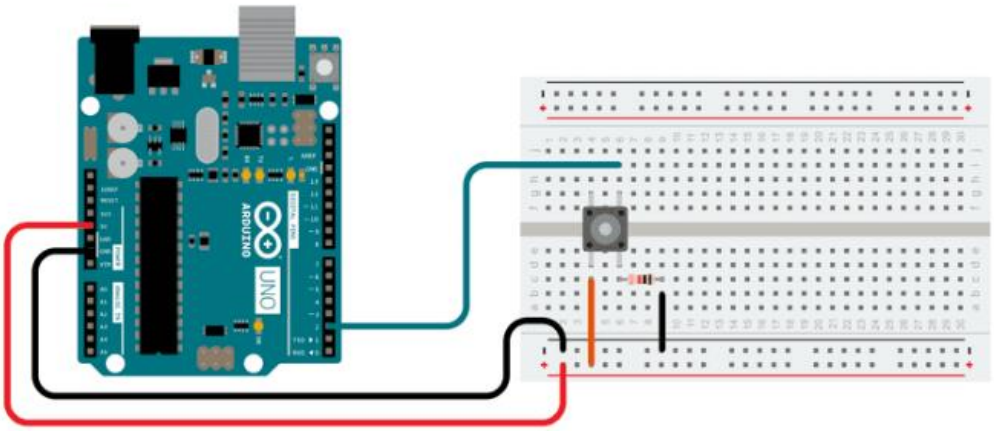
Il peut être mêlant d'utiliser un bouton-poussoir. On se demande tout le temps comment ça se branche. Dans cet exemple, on va voir comment lire un bouton.

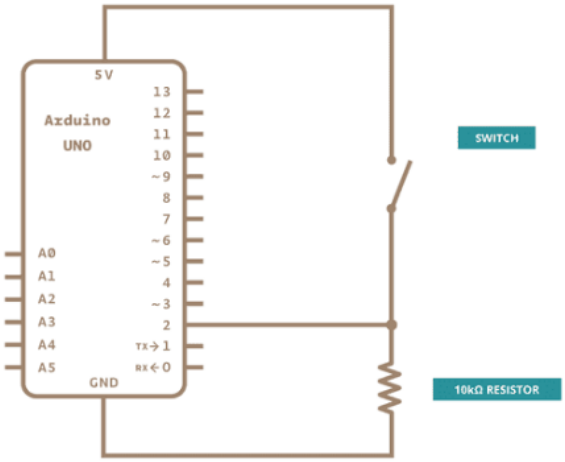
Sur le schéma suivant, les pattes colorées en vert sont connectées ensemble à l'interne. Les pattes surlignées en mauve aussi sont déjà connectées entre elles à l'interne. Quand on appuie sur le bouton, les pattes en vert se connectent avec les pattes en mauve.



Donc, on peut connecter un côté du bouton (ici gauche) au 5V et l'autre côté au GND. Ici, la résistance est mise du côté du GND. C'est donc une résistance de pull-down. Ça veut dire que lorsque tu n'appuies pas sur le bouton, l'Arduino va lire 0V. Lorsqu'on appuie le 5V se connecte et l'Arduino lit le 5V. On pourrait faire l'inverse et brancher la résistance du côté entre le bouton et le 5V à la place.

L'Arduino lit l'état du bouton via le fil turquoise, ici connecté à la patte 2 (digital). Mettre ce fil du côté de la résistance (si on le mettait de l'autre côté, on verrait toujours 5V).





Dans l'exemple de code, ils changent l'état d'une DEL lorsqu'on appuie sur le bouton. D'abord, ils déclarent en variables globales les numéros des pins utilisées. Cela peut être utile si on ne veut pas se souvenir que la pin 13 est pour la DEL. Au lieu, on peut juste utiliser la constante ledPin dans le code.

Ensuite, ils déclarent une variable pour lire l'état du bouton.

Dans le setup, ils initialisent la pin de la DEL en sortie (car on fournit du courant à la DEL) et la pin du bouton en entrée, car on lit le bouton.

Dans la loop, on lit l'état du bouton avec la fonction digitalRead. C'est comme digitalWrite, mais c'est pour lire une entrée, au lieu d'écrire sur une sortie. Digital = numérique, analog = analogique.

Si l'état du bouton est HIGH (appuyé car on a dit qu'il y a 5V dans ce montage lorsqu'on appuie sur le bouton), on allume la DEL. Sinon (bouton pas appuyé), on éteint la DEL. Pour allumer et éteindre la DEL, on utilise bien évidemment digitalWrite avec les paramètres LOW et HIGH.

```
// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2;  // the number of the pushbutton pin
const int ledPin = 13;    // the number of the LED pin

// variables will change:
int buttonState = 0;  // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Debouncing et autres pour un bouton-poussoir

Comme les profs l'ont mentionné, les boutons ont du rebond. Sur le PCB, on a déjà mis des filtres pour se débarrasser de ce problème, mais si vous voulez faire des tests, il y a un moyen de faire un debouncing de manière software (logicielle).

> Il est recommandé d'ajouter un filtre *anti-rebond* de type passif, RC passe-bas d'ordre 1. Ceci permet d'éviter les fausses lectures de double déclenchement du bouton, en cas de rebondissement du bouton. Certains boutons sont plus sujets au phénomène de rebondissement que d'autres, ou le deviennent en vieillissant. La constante de temps du filtre RC peut, généralement, être de l'ordre de 1 à 10ms.

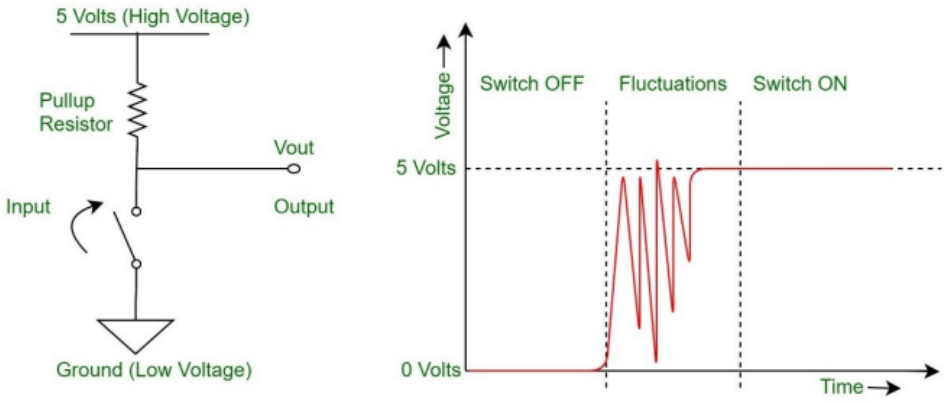


Figure 1 : Interrupteur sans filtre anti-rebond

Quand on détecte que le bouton est appuyé, on peut ajouter un délai de l'ordre de 50 ms ou moins pour ignorer les fluctuations générées. Par exemple, voici un exemple tiré du code d'Arthur.

```
if(digitalRead(22) == LOW)
{
  while(digitalRead(22) == LOW){delay(10);}
```

Si la pin 22 (un bouton sûrement) est à LOW (appuyé sûrement), tant que la pin 22 est à LOW (tant que le bouton est appuyé), on applique un délai de 10 ms. Cela empêche l'utilisateur de laisser appuyé son bouton longtemps et que ça compte comme plusieurs appuis séparés.