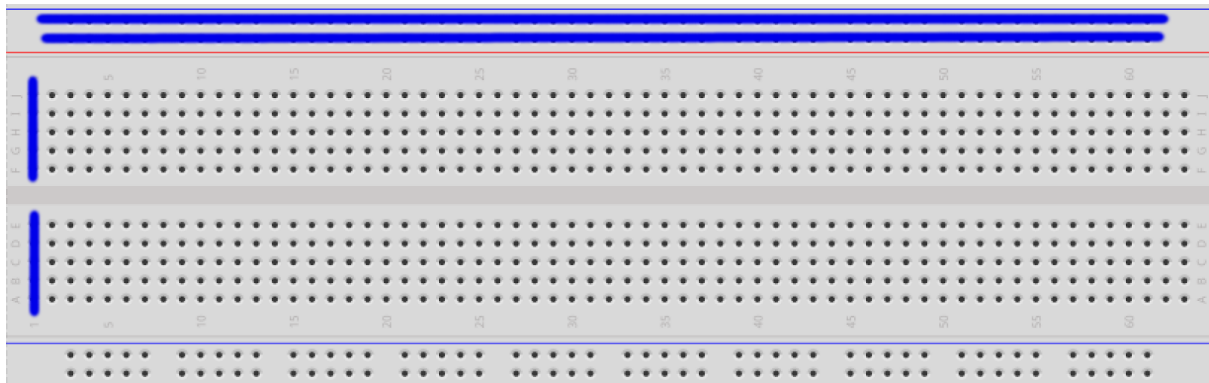


Prise en main du matériel de prototypage

Montage d'un circuit simple sur platine d'essai

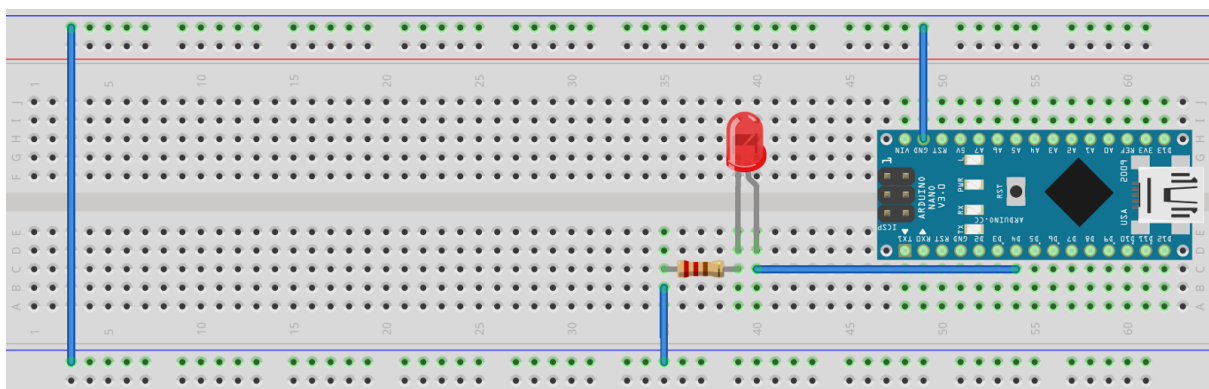
Une plaque de prototypage rapide, aussi appelée platine d'essai ou breadboard, permet de créer des circuits électroniques sans soudures. L'électricité la parcourt de manière horizontale et verticale, comme l'indique les traits bleus sur le schéma ci-dessous.



Sur cette platine d'essai, nous allons prendre en main nos composants électroniques, le contrôleur Arduino, les LEDS et tester nos premiers programmes.

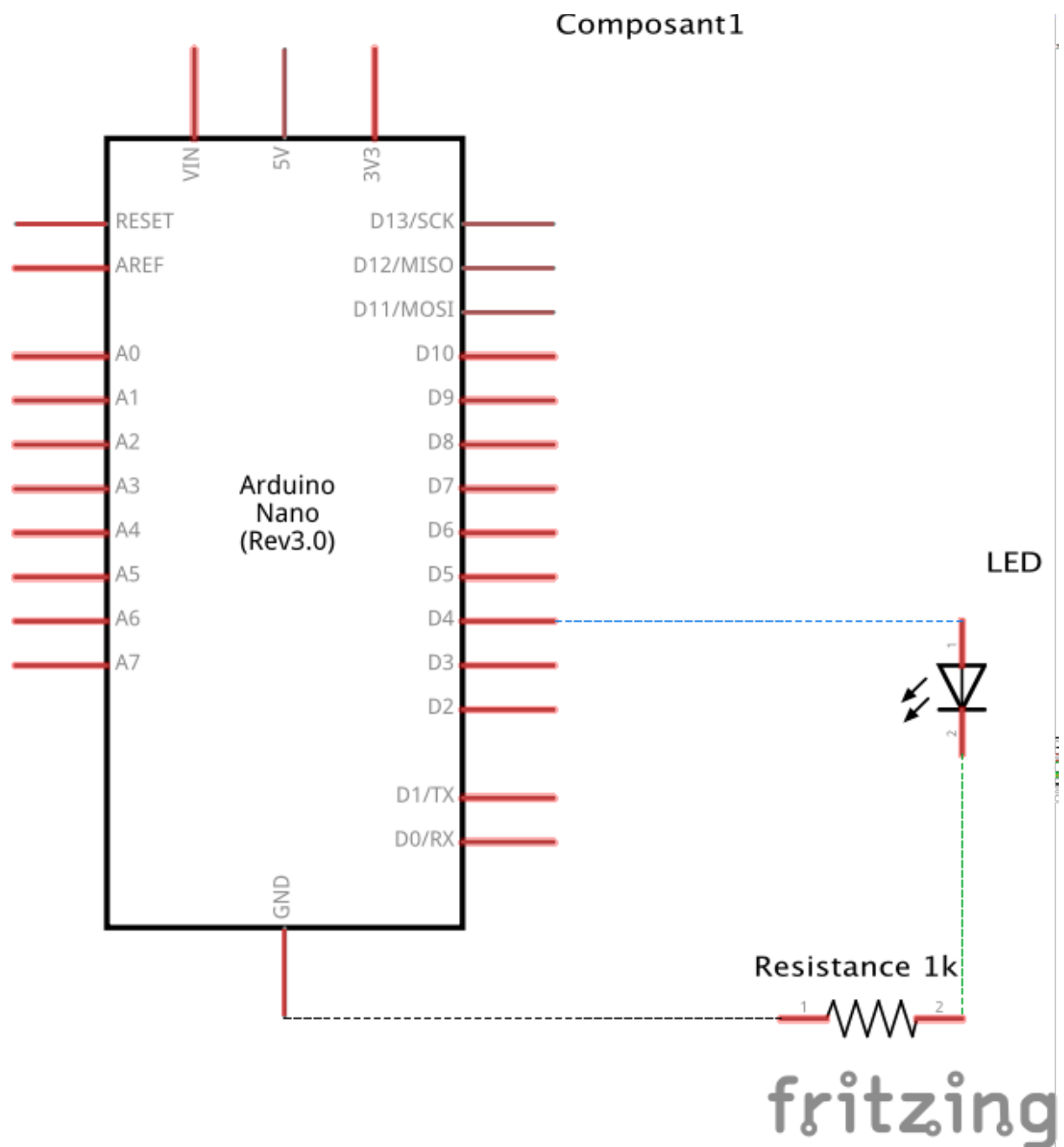
Bases de programmation et clignotement d'une LED

Dans un premier temps, nous allons connecter une LED à notre contrôleur sur la platine d'essai, afin de tester un programme de clignotement.



fritzing

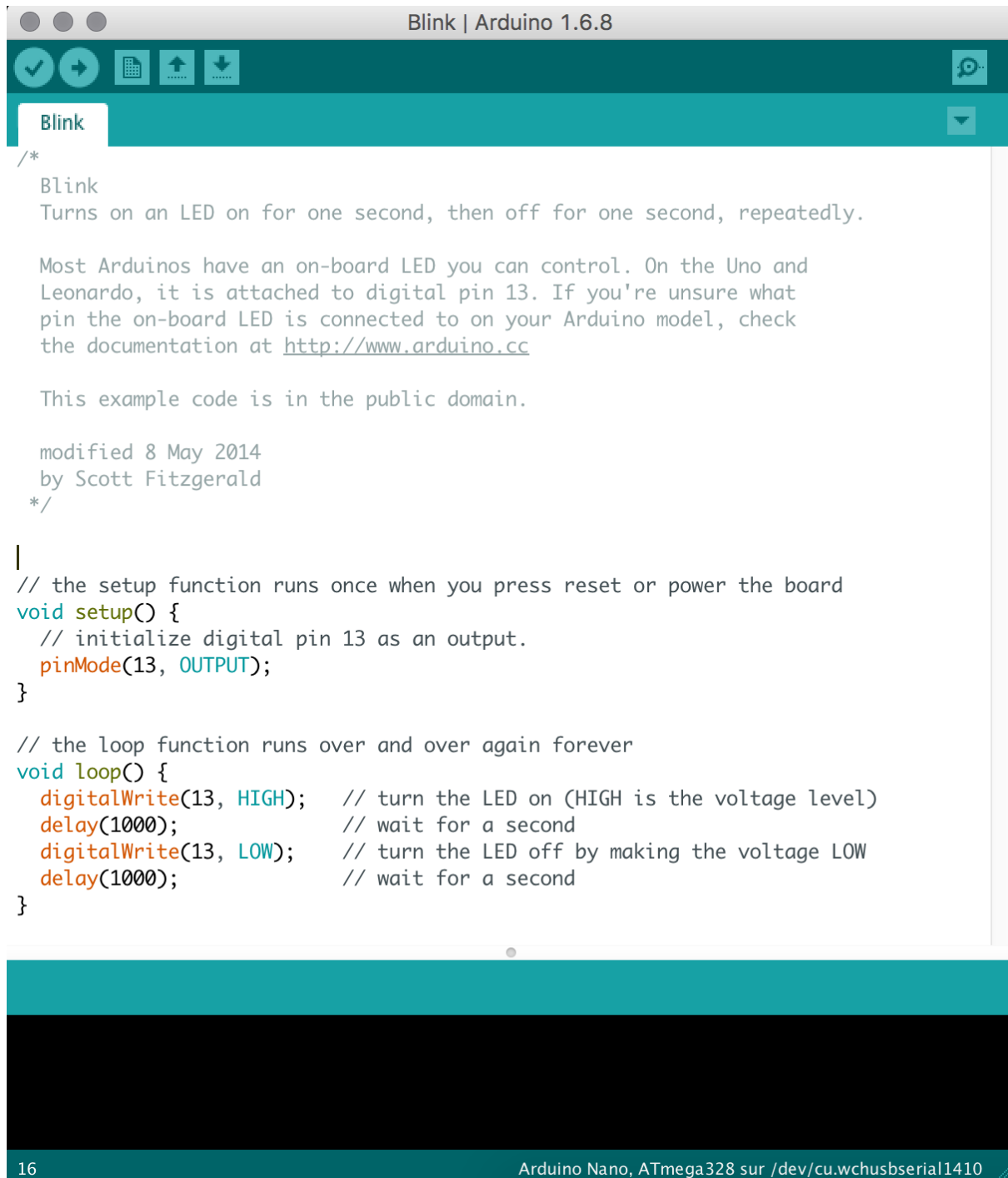
La représentation schématique de ce circuit est la suivante :



A présent, nous pouvons passer à l'étape de programmation du contrôleur via l'environnement de développement Arduino sur notre ordinateur.

Le récapitulatif des boutons de l'interface se trouve dans les annexes.

En cliquant sur "Fichiers" puis "Exemples" > "01.Basics" > "Blink", la fenêtre de code suivante s'ouvre :



Compréhension du code :

```
void setup() {  
  
}
```

```
void loop() {  
  
}
```

Tout programme Arduino est composé de deux fonctions principales. Les fonctions sont des parties du programme qui exécutent des commandes spécifiques. Elles ont un nom unique qui est appelé quand on en a besoin. Les deux fonctions obligatoires à un programme Arduino sont setup() et loop(). Elles sont déclarées, écrites, comme indiqué dans l'image ci-contre.

La fonction setup() est chargée qu'une seule fois, lors du démarrage du contrôleur. Elle permet de lui indiquer quels sont les composants présents sur le circuit, et sur quel pin ils sont connectés. Dans le code d'exemple, nous avons :

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

La fonction pinMode() indique que nous avons un composant sur le pin 13, réglé sur OUTPUT, c'est à dire que c'est une sortie, telle qu'une LED.

Dans notre schéma, nous constatons que l'Arduino Nano ne comprend pas de pin 13 et que la LED est connectée au pin 4. Nous pouvons donc éditer le code de la manière suivante :

```
void setup() {  
    pinMode(4, OUTPUT);  
}
```

La fonction loop() est chargée de manière continue, c'est à dire que les commandes qu'elle contient sont appelés sans arrêt, tant que le contrôleur est sous tension. C'est une boucle.

Dans ce premier exercice, nous voulons changer le voltage envoyé à la LED de manière régulière. Pour cela, nous utilisons la fonction digitalWrite() qui nous permettra d'envoyer un voltage de 0V ou 5V à un pin de sortie. Elle est composée de deux paramètres : le pin que nous voulons cibler et l'état que nous voulons lui donner (0V=LOW ou 5V=HIGH).

Elle est suivie de la fonction delay() qui va nous permettre de temporiser le clignotement. En effet, delay() indique au contrôleur qu'il doit attendre X temps avant d'exécuter la prochaine commande. La valeur attendue en paramètre de cette fonction correspond au nombre de millisecondes que doit attendre le contrôleur, soit 1 seconde = 1000.

Dans le code d'exemple, nous avons :

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

Cela indique que le programme va allumer la LED, sur le pin 13, attendre une seconde, puis l'éteindre et attendre une nouvelle seconde avant de recommencer les commandes de manière infinie, tant que le contrôleur est sous tension. Pour que ce code s'ajuste bien à notre circuit, il faut simplement que nous changions la valeur du pin ciblé par digitalWrite() puisque notre LED est sur le pin 4. Ce qui donne :

```
void loop() {
  digitalWrite(4, HIGH);
  delay(1000);
  digitalWrite(4, LOW);
  delay(1000);
}
```

Pour faciliter la compréhension de votre code lors de sa lecture sur votre ordinateur, vous pouvez rédiger des commentaires, qui n'auront aucune incidence sur le programme, en mettant // devant votre texte, tel que :

```
// ceci est un commentaire sur une seule ligne
ou /* votre texte */ :
/* ceci est un
commentaire sur
plusieurs lignes */
```

Pour envoyer, ou plutôt téléverser votre code sur votre contrôleur, branchez celui-ci sur votre ordinateur grâce au câble USB.


Ensuite il faut s'assurer que le logiciel est bien configuré pour reconnaître l'Arduino Nano :

Dans "Outils" > "Type de carte", sélectionner Arduino Nano.

Dans "Outils" > "Port", vérifier que le port série est correct, il devrait être renseigné automatiquement lors du branchement du contrôleur à l'ordinateur, tel que :

Windows	MacOS / Linux
---------	---------------

COM X	dev/tty.usbmodem
-------	------------------

Une fois que la configuration est vérifiée, vous pouvez appuyer sur le bouton de téléversement , la console vous indiquera la progression et affichera "téléversement terminé" quand le programme sera bien enregistré sur la carte. Vous verrez alors votre LED clignoter.

Pour aller plus loin

A présent, nous allons aborder de nouvelles notions fondamentales de la programmation Arduino en modifiant le code utilisé précédemment.

Les variables

Une variable est une façon de nommer et de stocker une valeur pour une utilisation ultérieure par le programme, comme les données d'un capteur ou une valeur intermédiaire utilisée dans un calcul.

Déclarer des variables

Avant d'être utilisées, toutes les variables doivent être déclarées. La déclaration d'une variable signifie définir son type et, le cas échéant, définir une valeur initiale (initialiser la variable). Les variables ne doivent pas obligatoirement être initialisées (attribuées une valeur) lorsqu'elles sont déclarées, mais cela s'avère souvent utile.

Par exemple, dans le programme d'exemple pour le clignotement de LED, nous pouvons améliorer le code en ajoutant la variable représentant le pin de la LED, afin d'éviter à avoir à répéter cette valeur plusieurs fois dans le programme, en l'occurrence, dans les fonctions `digitalWrite()` et `pinMode()`. Avec une variable déclarée une fois au début du programme, avant la fonction `setup()`, tel que :

```
int LEDpin = 4;
```

Nous utilisons ici le type `int`, qui représente un nombre entier. Chaque variable doit avoir un type défini lors de sa déclaration, la liste complète étant détaillée sur la documentation d'Arduino : <https://www.arduino.cc/en/Reference/VariableDeclaration>

Remplaçons maintenant toutes les paramètres de nos fonctions en mettant `LEDpin` à la place de la valeur 4, tel que :


```
int LEDpin = 4;

void setup() {
  pinMode(LEDpin, OUTPUT);
}

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

Nous pourrions ainsi décider de changer de pin en cours de route, changer une seule fois cette valeur en début de programme, sans avoir à modifier chaque fonction. C'est un gain de temps considérable sur de longs programmes et un réflexe de développement fondamental !

Moniteur série

Lorsque vous téléchargez un programme sur votre carte Arduino qui est connectée à votre ordinateur, vous pouvez ouvrir le moniteur série avec l'icône de la loupe , pendant qu'il tourne.

Il peut servir comme afficheur (l'Arduino envoie des informations qui sont traitées par votre ordinateur et affichées par le moniteur). Pour cela, vous devez initialiser la communication dans la fonction setup() en utilisant la commande Serial.begin(). Ensuite dans la fonction loop(), on utilise la commande Serial.println() pour afficher du texte, ou la valeur d'une variable comme dans l'exemple de la photorésistance ci-dessus.

Il peut aussi servir à envoyer des données à l'Arduino avec les fonctions Serial.available() et Serial.read().

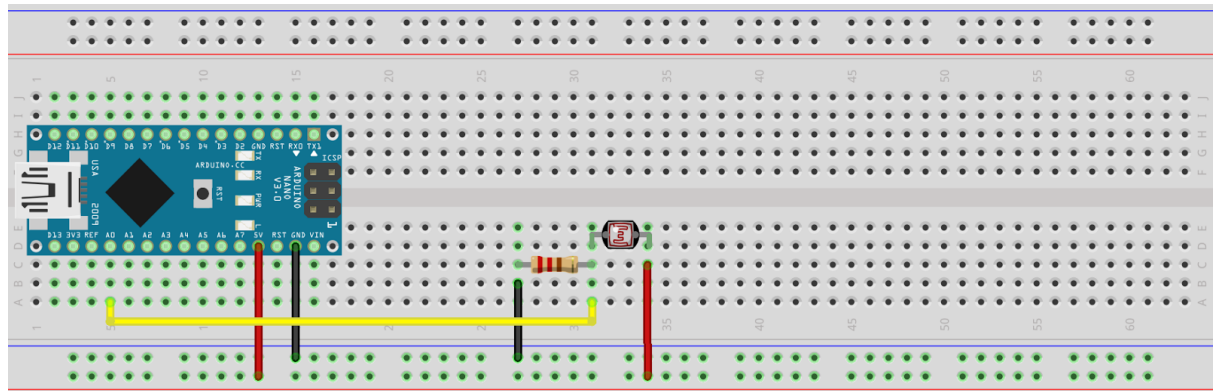
Récupération de données capteur simple

La photorésistance

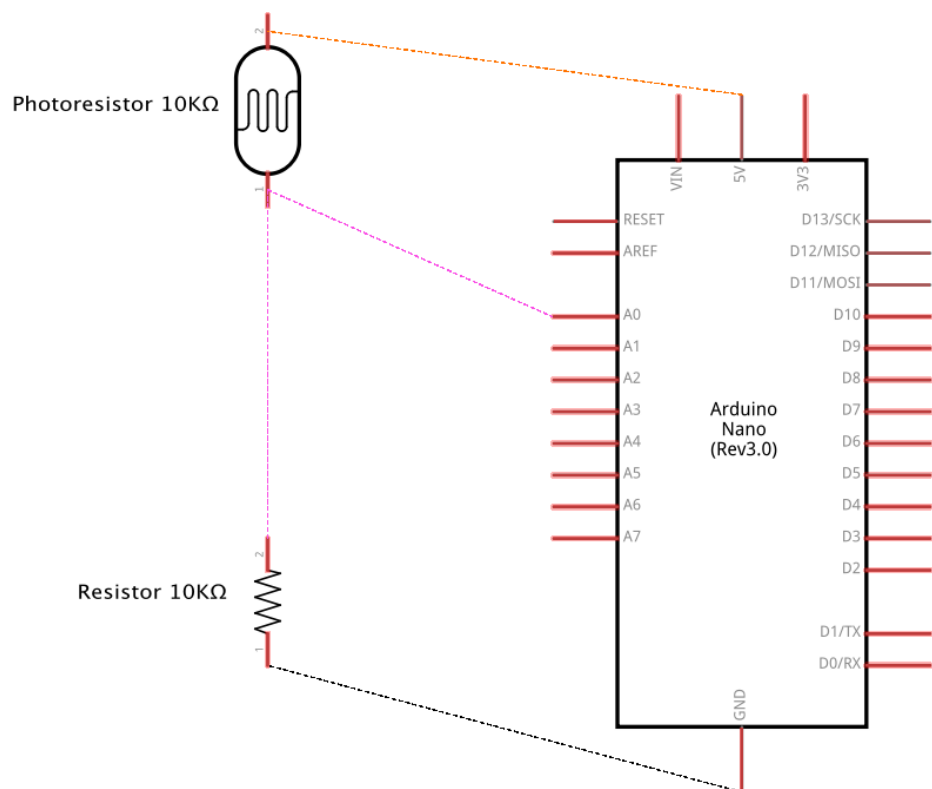
Les photorésistances sont des senseurs qui permettent de détecter la lumière. Les photorésistances sont des résistances dont la valeur résistive, en ohms Ω , change en fonction de la quantité de lumière qui atteint le capteur. Elles sont abordables, existent sous

de nombreux formats/tailles, disponibles sous de nombreuses spécifications mais sont très imprécises.

Montage



fritzing



fritzing

code


```
read-sensor

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

void loop() {
  int value = analogRead(A0);
  Serial.println(value);
  delay(250);
}

Enregistrement terminé.

Le croquis utilise 2 812 octets (9%) de l'espace de stockage de programmes. Le maximum
Les variables globales utilisent 200 octets (9%) de mémoire dynamique, ce qui laisse 1

11 Arduino Pro or Pro Mini, ATmega328 (5V, 16 MHz) sur /dev/cu.usbserial-A900awzr
```

Les conditions

C'est un choix que l'on fait entre plusieurs propositions. En informatique, les conditions servent à tester des variables. Cela va nous permettre d'attribuer des comportements différents suivant certaines valeurs. Par exemple, si le bouton est appuyé, on fait clignoter la led, s'il n'est pas appuyé elle est éteinte.

Une condition s'écrit de la manière suivante :

```
if { // contenu de la condition
  // instructions à exécuter
} else { // si la condition précédente est fausse
  // instructions à exécuter
}
```

Pour définir les paramètres qui constitue la condition, on utilise les symboles suivants :

Symbole	Utilité	Signification
==	Ce symbole, composé de deux égales, permet de tester l'égalité entre deux variables	... est égale à ...
<	Celui-ci teste l'infériorité d'une variable par rapport à une autre	...est inférieur à...

>	Là c'est la supériorité d'une variable par rapport à une autre	...est supérieur à...
<=	teste l'infériorité ou l'égalité d'une variable par rapport à une autre	...est inférieur ou égale à...
>=	teste la supériorité ou l'égalité d'une variable par rapport à une autre	...est supérieur ou égal à...
!=	teste la différence entre deux variables	...est différent de...