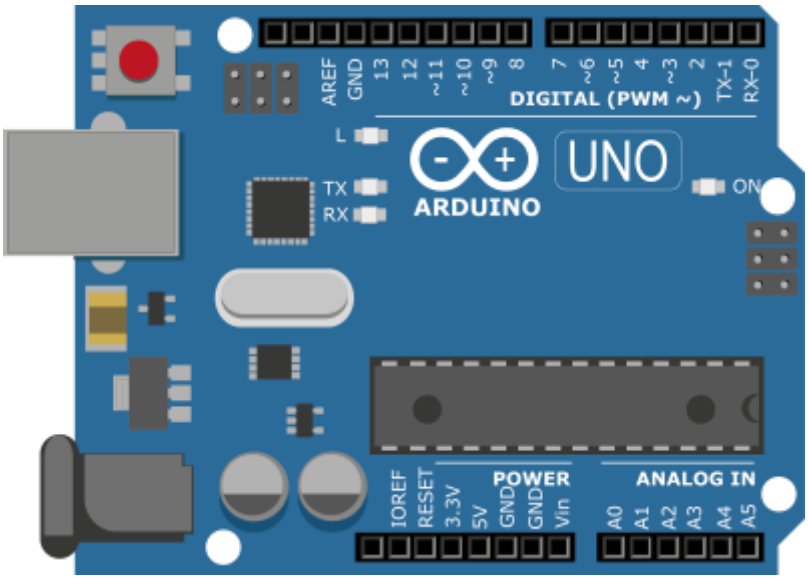
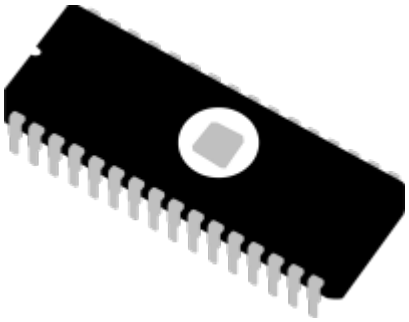
# Introduction à Arduino




## Microcontrôleur

### Présentation



Un microcontrôleur est un composant électronique qui regroupe les principaux composants d'un ordinateur.

Ils sont utilisés dans des systèmes électroniques et informatiques de taille limitée et spécialisée dans une tâche, ce sont les **systèmes embarqués**, où ils tiennent le rôle de “cerveau”, en traitant l'information et en permettant l'**automatisation** du système grâce à un programme implémenté.

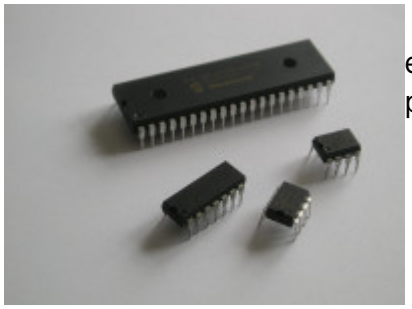
Un microcontrôleur n'est pas aussi puissant qu'un micro-ordinateur (un ordinateur personnel). Il est destiné à réaliser une tâche unique/répétitive plutôt simple. Sa faible puissance de calcul permet une faible consommation d'énergie, sa taille réduite permet une incorporation dans de nombreux systèmes.

## Composition

Un microcontrôleur est composé (principalement) de:

- **Horloge** : Elle permet d'accéder à la notion de temps.
- **Mémoire volatile** : (RAM) Elle stock les données temporaires comme les variables de calculs.
- **Mémoire morte** : (ROM) Elle stock les données qui ne sont pas amenées à changer, comme le programme de mise sous tension de la carte ou encore le programme implémenté dans la carte.
- **Bornes entrées/sorties** : Réception ou Envois des données.

## Périphériques



Les microcontrôleurs sont implémentés sur une carte électronique qui est reliée à des périphériques. Ces périphériques peuvent être des périphériques de capture ou chargé d'exécuter une action.

*Par exemple un thermomètre capture l'information liée à la température, un moteur peut être chargé de l'action "tourner" afin d'actionner les palles d'un ventilateur.*

Le microcontrôleur est chargé de faire le lien entre informations entrantes et informations sortantes en fonction du **programme** qui lui a été donné d'exécuter.

*Par exemple: en fonction de la température détectée par le thermomètre, le microcontrôleur est chargé de commander au moteur de s'allumer ou de s'éteindre.*

La notion de **transport de l'information** est importante, elle est réalisée par le **BUS**.

## Programmation

Un microcontrôleur est programmable. C'est-à-dire qu'un programme ou algorithme peut être implémenté. L'algorithme écrit dans le langage de programmation utilisé est d'abord transcrit dans un langage que comprend le microcontrôleur puis enregistré dans la ROM. L'algorithme est ensuite exécuté.

Pour programmer une carte Arduino il est nécessaire d'écrire un algorithme dans un langage particulier. Ce langage est le **C++** avec des bibliothèques de fonctions et de classes déjà importées et destinées à faciliter la programmation.

Une fois écrit via l'**IDE Arduino**, l'algorithme est compilé et transféré à la ROM de la carte (tout ceci de manière automatique via l'**IDE**).

La compilation est l'action de transcription d'un algorithme écrit en langage de

programmation (compréhensible par l'humain) en un langage directement compréhensible et exécutable par la machine.

Les langages Python et Matlab ne sont pas compilables (sauf manipulation spéciale), ils nécessitent un système plus complexe fonctionnant “derrière eux” : un interpréteur (ce sont des langages interprétés), hormis les avantages de cette technique, l'interprétation impacte de manière négative les performances. Les langages C, C++, Fortran sont des langages dont le code une fois compilé est un fichier qui peut être directement exécuté par le processeur, hormis les inconvénients, la compilation permet de bonne performances.

Ainsi les microcontrôleurs ne possèdent pas de systèmes d'exploitations, ce qui leur permet une efficacité relative dans l’exécution des algorithmes, bien que leurs ressources soient limitées.

## Fonctionnement

### Exécution de l'algorithme

L’exécution de l'algorithme est réalisée séquentiellement, c'est-à-dire que les instructions sont réalisées l'une après l'autre. Chaque instruction attendant que la précédente soit achevée pour être exécutée (sauf indications spéciales dans le code).

L’exécution d'une instruction est réalisée en plusieurs étapes, qui sont en général :

- **Lire** en mémoire l'instruction à exécuter
- **Analyser** l'instruction pour déterminer quelles opérations réaliser
- **Exécuter** l'instruction
- **Écrire** en mémoire les résultats de l'instruction

Les parties lire en mémoire et écrire en mémoire nécessite un **adressage des données**, une sorte de carte où les coordonnées dans la mémoire de chaque donnée est écrite.

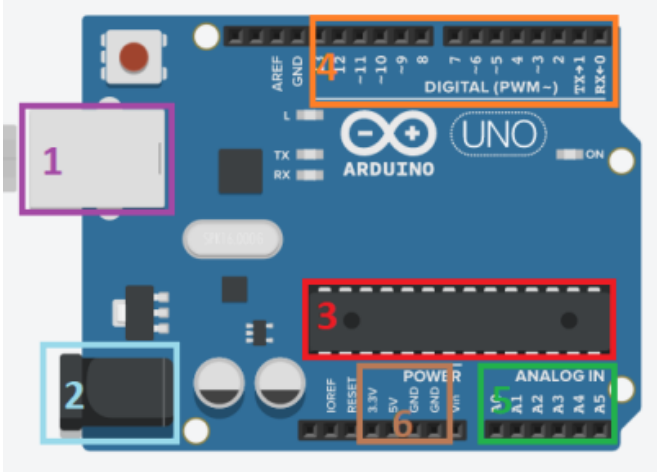
### Cycle d'horloge

Le cycle d'horloge est la période du signal d'horloge, qui est un signal périodique oscillant qui rythme les actions du processeur.

L'Instruction par cycle d'horloge est une grandeur qui indique la performance du processeur, elle compte le nombre d'instructions exécutées pendant un cycle d'horloge.

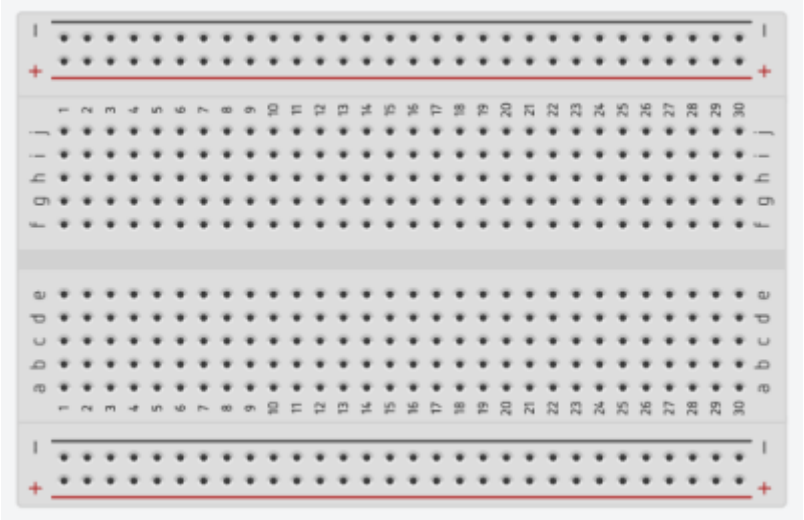
# Carte Arduino : Premiers pas

## Carte Arduino

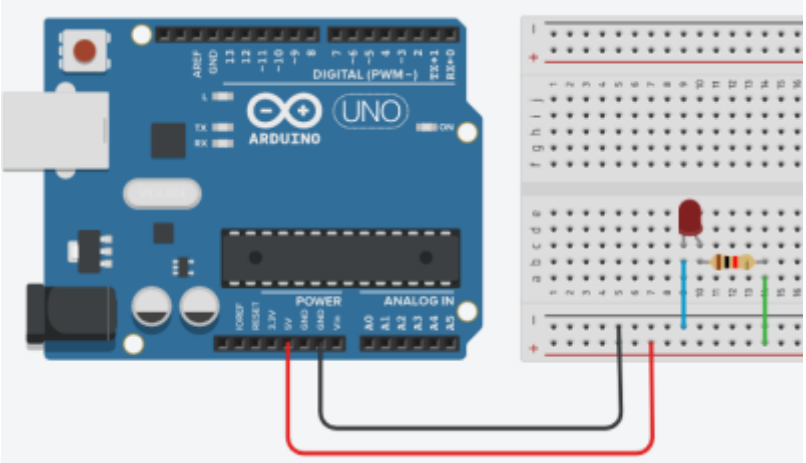



1. Port USB
2. Port d'alimentation
3. Microcontrôleur
4. Pins digitaux
5. Pins analogiques
6. Sortie de tension et masses (GND)

## Breadboard



On se sert d'une breadboard pour faire des circuits sans avoir besoin de souder les composants, ce qui est pratique pour l'apprentissage le prototypage.



Dans cet exemple, la résistance est reliée à l'alimentation grâce aux fils rouge et vert (**lien horizontale**) alors que la résistance et la LED sont liées grâce à un **lien vertical** (encadré orange).

# IDE Arduino

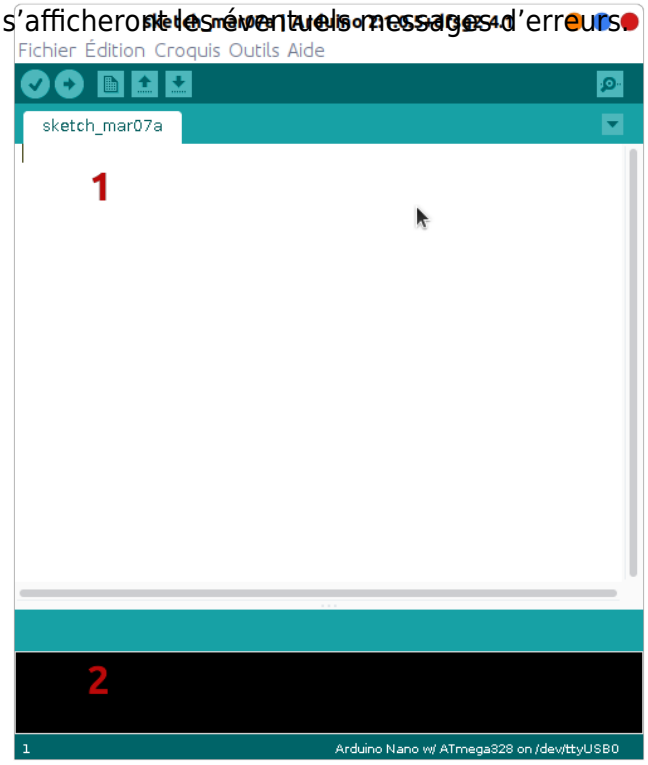
> Un IDE (Integrated Developpement Environment) est un logiciel qui fournit des outils facilitant le développement, la programmation de programme ou d'application.

L'IDE d'Arduino est l'un des moyen les plus simples et le plus pratiques pour développer un programme puis le **téléverser** (l'enregistrer sur la carte). Il automatise de nombreuses tâches et facilite la communication avec la carte.

Il est disponible gratuitement sur ce lien: Software Arduino

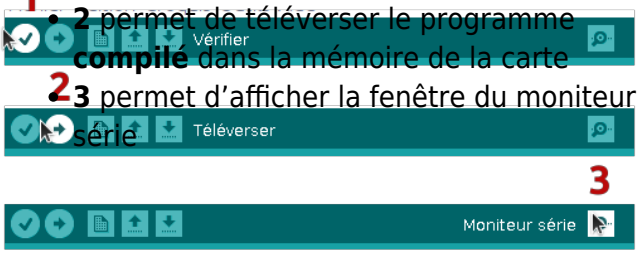
## Interface de l'éditeur

La zone **2** est le **terminal**, c'est ici que s'afficheront les éventuels messages d'erreurs.




La zone **1** est la zone de saisie. C'est ici que le code devra être typer.

- **1** permet de vérifier le code et détecter les éventuelles erreurs
- **2** permet de téléverser le programme **compilé** dans la mémoire de la carte
- **3** permet d'afficher la fenêtre du moniteur série




Le bouton:

## Moniteur Série

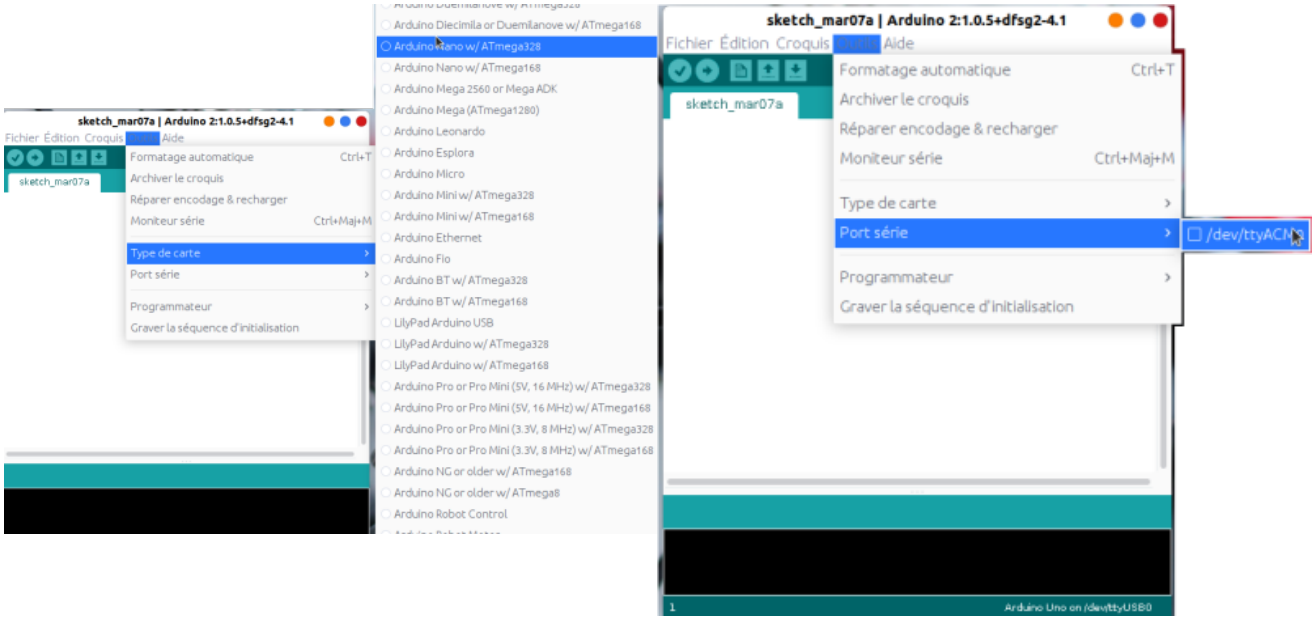
Le moniteur série permet d'afficher des données, des résultats, mais aussi d'envoyer des données à l'Arduino.

## Connexion d'une carte

Voici comment connecter une carte à votre ordinateur.

- Brancher la carte à votre ordinateur via le
- Indiquer le port USB de communication de votre ordinateur sur lequel est branché le câble :
  Outils > Port série > "votre_port"

câble USB
- Ouvrez l'IDE
- Indiquer la carte qui a été branchée (le nom est inscrit sur la carte) :
  Outils > Type de carte > "votre_carte"
  - Il peut y avoir plusieurs ports (essayer un par un)
  - Le nom des ports varie en fonction des systèmes d'exploitations Windows ou OSX/Linux




# Code

## Code de base

Le langage utilisé avec Arduino est le langage **C++**, avec des bibliothèques importées par défaut. Ceux ayant déjà codé en C++ ou C ne seront donc pas dépaysés.

Ces langages induisent des principes que ceux n'ayant utilisé que python peuvent ne pas connaître, nous en détaillons quelques-uns.

- Déclaration des variables : Chaque variable doit être **déclarée** avant d'être utilisée. Ceci revient à dire quelles données **type** elles contiendrons et quelle taille de mémoire leur sera allouée. Deux types utilisés sont *int* pour un entier ou *double* pour un nombre à virgule.

```
int a;          //déclaration de la variable a de type int
a = 10;         //attribution de la valeur 10 à la variable a
double b;       //déclaration de la variable b de type double
b = 10.05;      //attribution de la valeur 10.05 à b
int c = 5;      //forme contracté de déclaration & attribution
```

- Chaque instruction doit se terminer par un

```
;
```

- L'indentation n'est pas obligatoire (on peut tout écrire sur une seule ligne).
- Ligne de commentaire

```
// exemple
```

- Bloc de commentaire

```
/* exemple */
```

- Les fonctions de bases sont les suivantes:
  - pour k allant de 0 à n (exclu)

```
for(int k = 0; k < n; k++){
    //bloc de code
}
```

  - si condition est vraie

```
if(condition == true){
  //bloc de code
}
```

  - tant que condition est vraie

```
while(condition == true){
  //bloc de code
}
```

## Code Arduino

Nous détaillons ici les fonctions de bases d'Arduino.

- **pinMode()** : Définir comme entrée **INPUT** ou comme sortie **OUTPUT** le pin (branchement) :

```
pinMode(5, INPUT);
pinMode(1, OUTPUT);

//ou en encore :
int nom_pin = 2;
pinMode(nom_pin, INPUT);
```

- **digitalRead()** : Lire la valeur binaire (soit **HIGH** soit **LOW**) du pin spécifié.

```
int valeur;
valeur = digitalRead(5);
```

- **digitalWrite()** : Envoyer la valeur binaire (soit **HIGH** soit **LOW**) au pin spécifié.

```
int nom_pin = 5;
int valeur = LOW;
digitalWrite(nom_pin, valeur);
```

- **analogRead()** : Lire la valeur analogique (de 0 à 1023 (dépendant des cartes) codée sur un type *int*) du pin spécifié.

```
int valeur;
valeur = analogRead(A5);
```

- **analogWrite()** : Envoyer la valeur analogique (de 0 à 1023 codée sur un type *int*) au pin spécifié.

```
int valeur;
valeur = 500;
int nom_pin;
analogWrite(nom_pin, valeur);
```

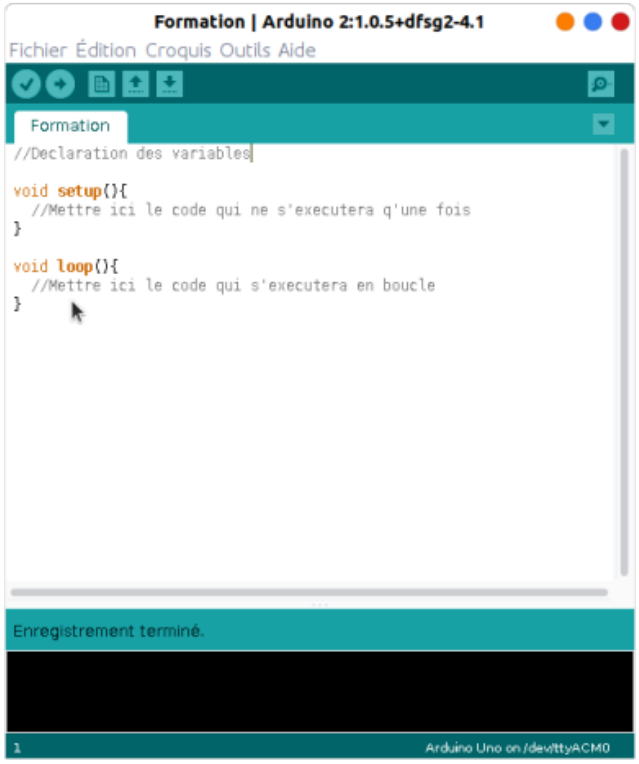
- **delay()** : Mettre en “pause” l’exécution du programme durant la durée en ms spécifiée.

```
delay(3000); //pause de 3s
```

#### Liste exhaustive des fonctions Arduino

Le site d'Arduino fourni une liste des fonctions disponibles et des détails les concernant : Arduino Reference

## Structure du programme




#### Déclaration des variables

Il se fait avant toute chose. on y insérer les différent nom de pin en gardant des noms caractéristiques.

Par exemple :

```
int Led = 2;
```

#### Initialisation

Elle se fait durant la bloc **setup()**, on y insère tous les **digitalMode()** afin de définir comment seront utilisés les différents pin.

#### Boucle d'exécution

Le bloc de code **loop()** est l'endroit où insérer votre code afin qu'il soit exécuté en boucle...

### Baud

Le baud concerne la commande énigmatique **Serial.begin(9600);**. Cette commande précise la vitesse de communication entre la carte et l'ordinateur. Il est nécessaire de la préciser si plus tard dans votre projet vous souhaitez utiliser le moniteur série (voir partie IDE) qui permet par exemple d'afficher en temps réel les données des capteurs de la carte.

## Applications

### Une LED...

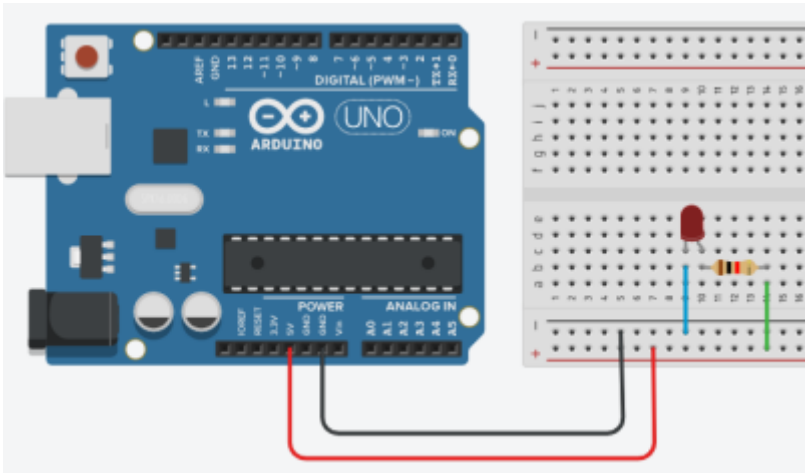
#### Matériel

- une LED
- une résistance de 10 k
- une breadboard
- des cables

La catode d'une **LED** ou **diode électroluminescente** est sa plus longue pate.

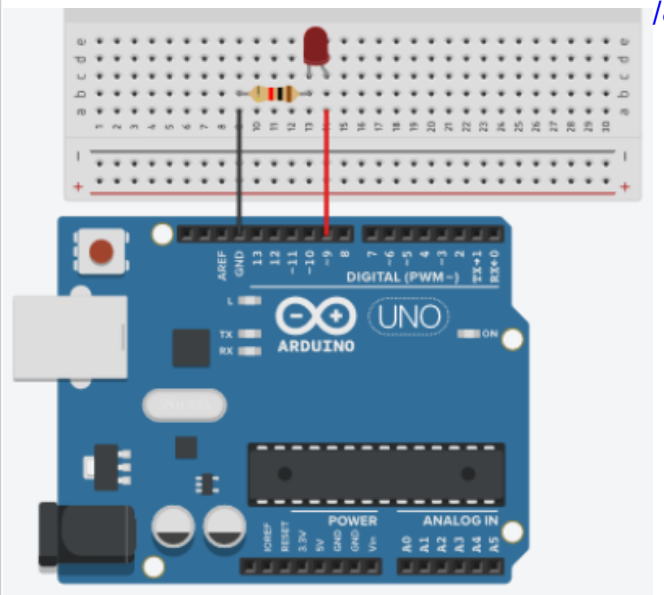
#### Enoncé

Allumer une LED avec une carte Arduino.




#### Enoncé Bis

Faire clignoter une LED avec une carte Arduino.




/applications/clignotement.cpp

```
int pin_led = 9;

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

void loop()
{
  digitalWrite(pin_led, HIGH);
  delay(3000);
  digitalWrite(pin_led, LOW);
  delay(3000);
}
```

## ... avec un bouton poussoir

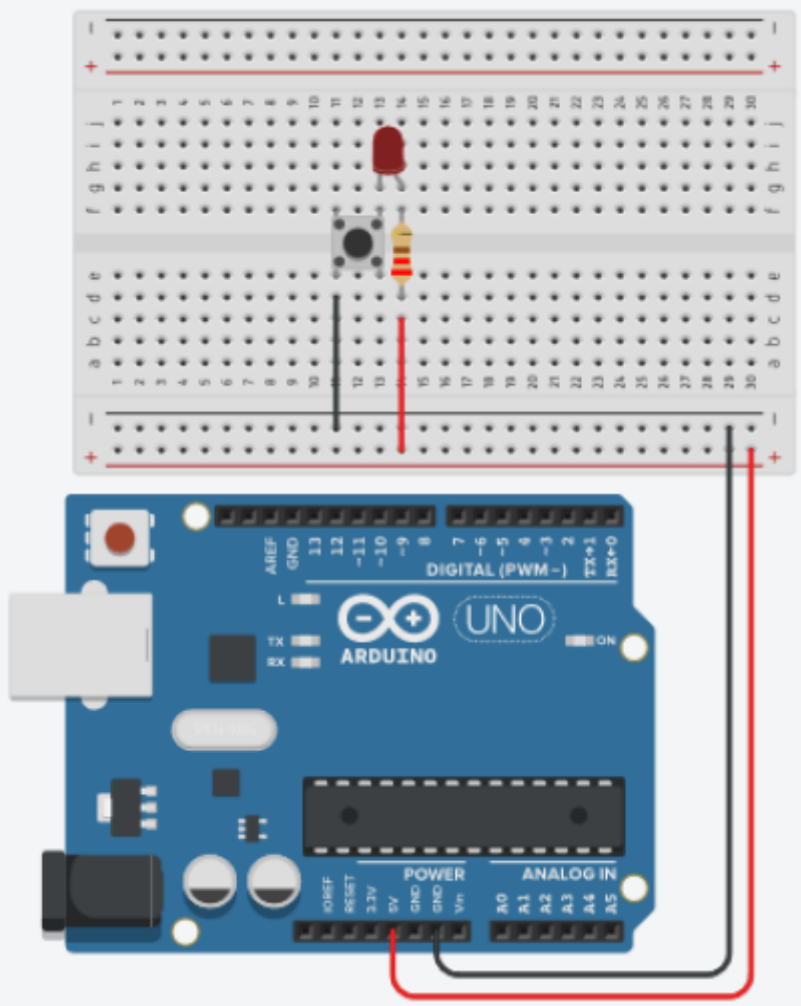
### Matériel

- une LED
- un bouton poussoir
- deux résistances de 10 k
- une breadboard
- des cables

Même si un **bouton poussoir** a quatre pattes, il reste un dipôle. Les pattes sont reliées deux à deux.

### Enoncé

Allumer la LED si le bouton poussoir est activé.




### Enoncé Bis

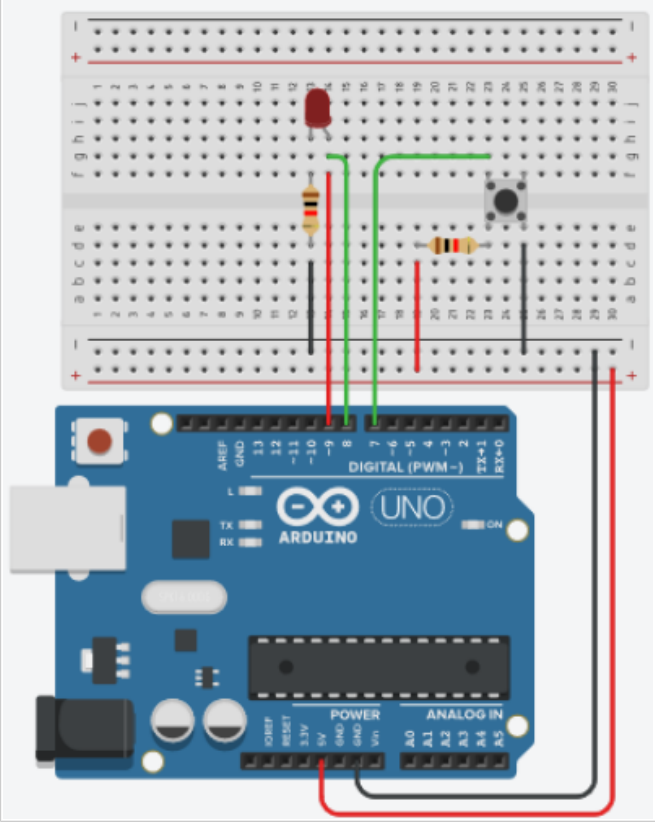
Allumer ou éteindre la LED lorsqu'on appuie sur le bouton poussoir.

/applications/bouton.cpp

```
int pin_led = 9;
int pin_etat = 8;
int pin_bouton = 7;

void setup()
{
  pinMode(pin_led, OUTPUT);
  pinMode(pin_etat, INPUT);
  pinMode(pin_bouton, INPUT);
}

void loop()
{
  if(digitalRead(pin_bouton) ==
LOW){
    if(digitalRead(pin_etat) ==
```




```
LOW){
      digitalWrite(pin_led,
HIGH);
    }
    else{
      digitalWrite(pin_led,
LOW);
    }
  }
}
```

## ... avec une photorésistance

### Matériel

- une LED
- une photorésistance
- deux résistances de 10 k
- une breadboard
- des cables

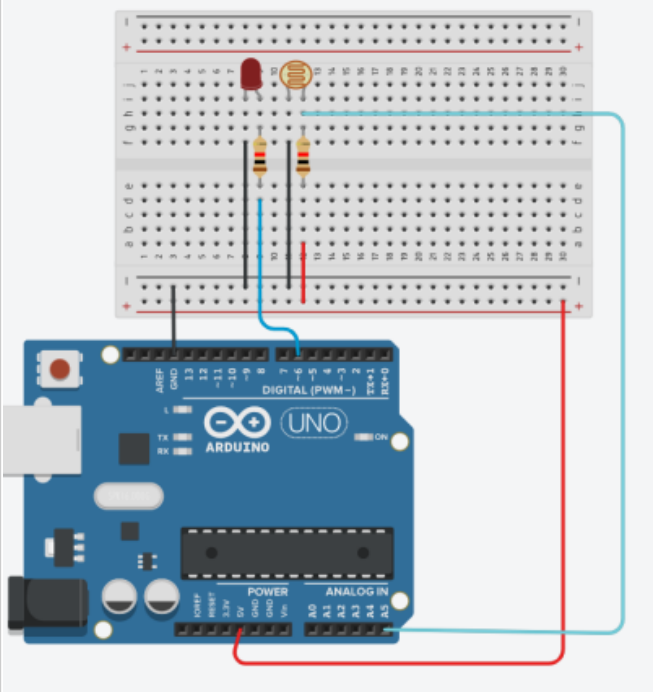
Plus la photorésistance reçoit de la lumière, plus la résistance est faible. Elle renvoie à la carte Arduino une valeur entre 0 et 1023.

### Enoncé

Si la valeur de la photorésistance est supérieure à 500, allumez la LED. Sinon éteignez-la.

/applications/photoresistance.cpp

```
int pin_photo = A5;
int pin_led = 6;
float val_res;
```




```
void setup() {
  pinMode(pin_photo, INPUT);
  pinMode(pin_led, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  val_res =
analogRead(pin_photo);
  if (val_res >= 500) {
    digitalWrite(pin_led,
HIGH);
  }
  else {
    digitalWrite(pin_led, LOW);
  }
  delay(1000);
}
```

## Ce qu'il faut retenir...

formation_1.mp4