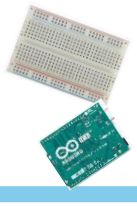
Étape 1 : BreadBoard et Arduino



Objectifs:

- ❑ Comprendre le fonctionnement d'une breadboard;
- ❑ Comprendre le fonctionnement d'une carte Arduino;
- ❑ Connecter une carte Arduino et une breadboard.

Comprendre l'électricité et les circuits électriques

On peut assimiler le courant dans un fil à de l'eau qui circule dans un canal : Le **courant** représente le débit volumique d'eau. Les **résistances** représentent des obstacles qui ralentissent le débit d'eau. Le **potentiel** représente la quantité d'eau disponible.
Le courant électrique, comme l'eau, cherche toujours à prendre le chemin avec le moins de résistance (avec le moins d'obstacles).
Il existe deux possibilités pour faire **diminuer le courant** :

- On le **ralentit à l'aide de résistances** : le débit diminue car la vitesse diminue (**Fig. 1**).
- On le **sépare en plusieurs branches** : le débit diminue car le volume dans chaque branche est divisé. (**Fig. 2**).

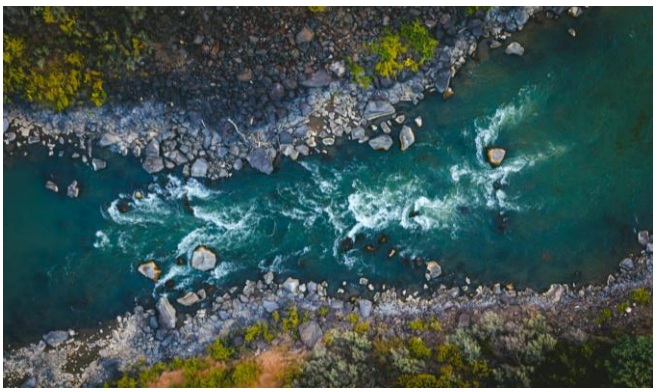


Fig. 1 : Rivière avec obstacles. Comme en électricité, la rivière rencontre des obstacles, et la vitesse de l'eau diminue même si c'est toujours le même volume d'eau qui circule. Donc le débit diminue.

Fig. 2 : Bifurcation d'une rivière. Comme en électricité, la rivière se sépare en deux, chaque branche garde la même vitesse qu'en amont, mais le volume d'eau circulant a diminué. Donc le débit diminue.

Illustration à l'aide des circuits en série et en parallèle

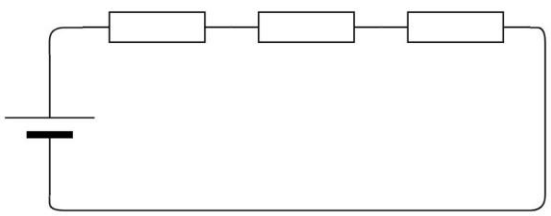


Fig. 3 : Schéma d'un circuit en série

Ici, les trois résistances sont en **série** . Toute l'électricité passe par la même branche où l'on a placé beaucoup de résistances. Donc **le courant diminue car on le ralentit**.

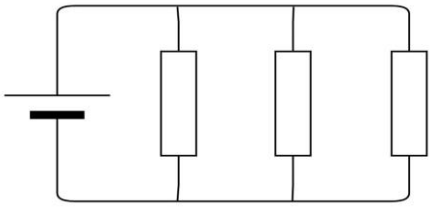


Fig. 4 : Schéma d'un circuit en parallèle

Ici, les résistances sont en **parallèle**. L'électricité se sépare en trois branches dotées d'une seule résistance chacune. Donc **le courant est plus faible dans chaque branche avec une résistance que dans la branche avec le générateur**

Maintenant que l'on connait mieux les bases de l'électricité et des circuits électriques, nous allons voir **comment mettre en œuvre les circuits électriques sur une breadboard**.

Comprendre le fonctionnement de la breadboard

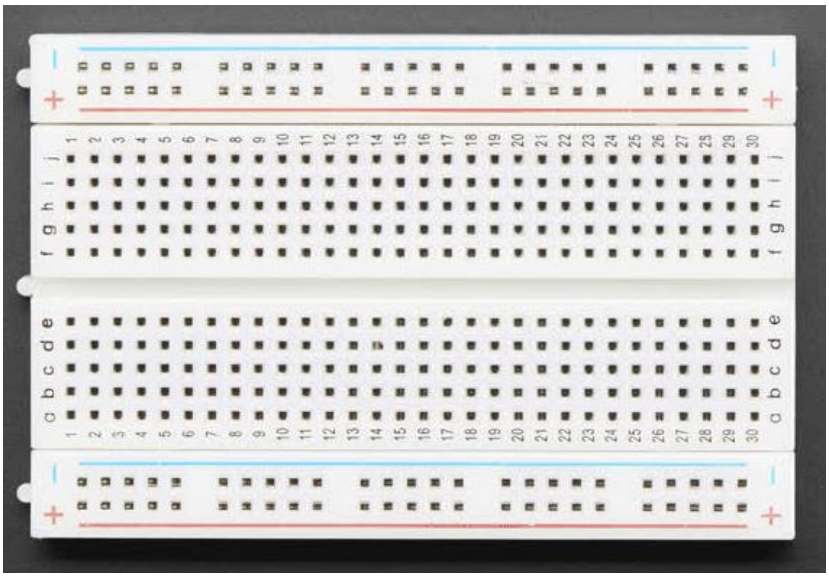


Fig. 5 : Photographie d'une breadboard

Les petits **trous** que l'on voit partout sur la breadboard (**Fig. 6**) sont des **points de connexion**. Ainsi, lorsque l'on formera notre circuit électrique, on branchera les fils et les composants dans les trous.

ATTENTION : Le passage suivant explique la logique des points de connexion sur la breadboard. C'est un point fondamental pour comprendre comment brancher correctement nos circuits électriques.

Sur la breadboard, les points de connexion ne sont pas tous indépendants. Certains correspondent au **même point.** Par exemple, chacune des lignes bleues de la **Fig. 6** regroupent des points de connexion qui correspondent à **un seul** point de connexion.
Sur la partie centrale de la breadboard, les "groupes" de points de connexion qui correspondent à un seul point de connexion sont les lignes verticales bleues (**Fig. 6**). Sur les parties supérieure et inférieure de la breadboard (au-dessus et en-dessous de la partie centrale sur le schéma **Fig. 6**), ces "groupes" sont les lignes horizontales bleues (**Fig. 6**).

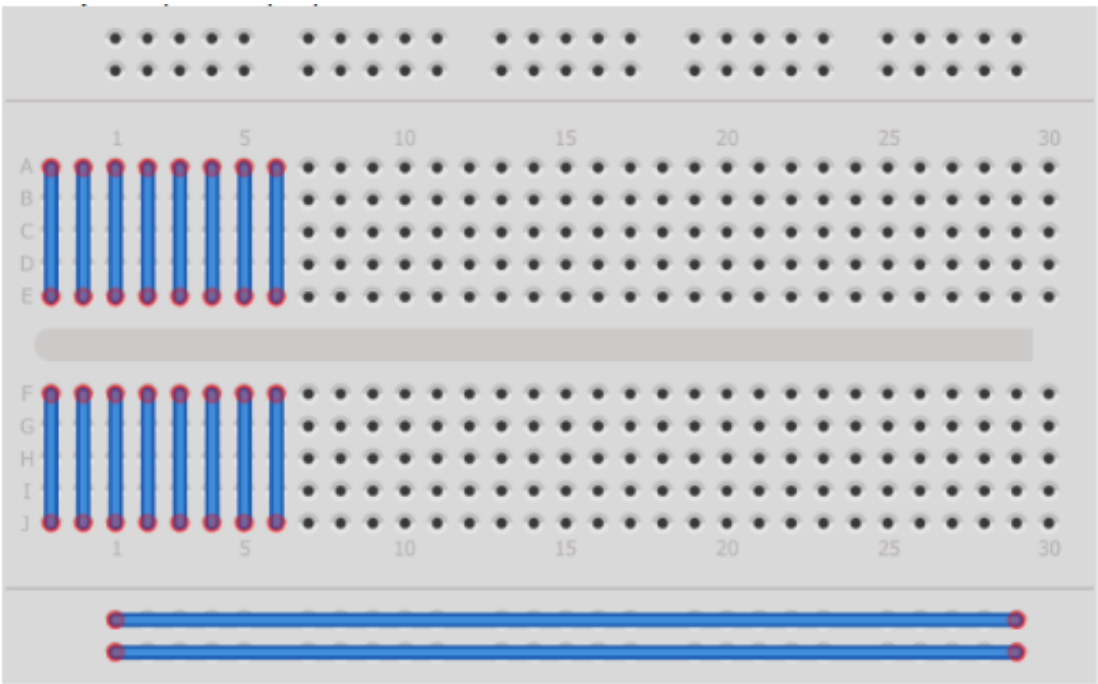


Fig. 6 : Schéma d'une breadboard

Mais pourquoi vouloir regrouper ainsi des points de connexion ?

Nous allons répondre à cette question par un exemple. Imaginons que l'on veut brancher le circuit de la **Fig. 7** :

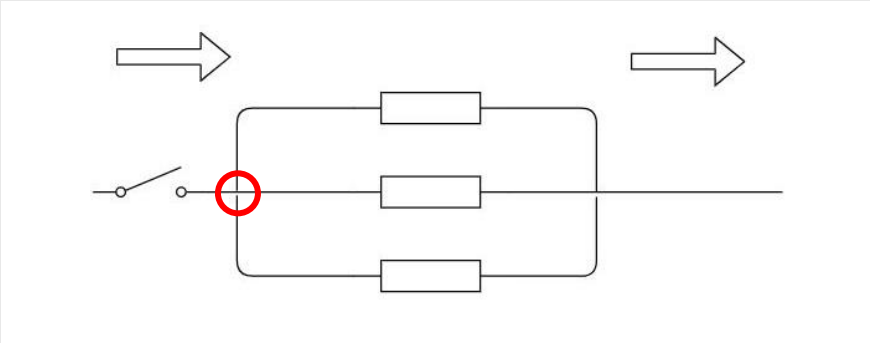


Fig. 7 : Schéma électrique avec trois résistances en parallèle

Au niveau du point rouge (**Fig. 7**), nous sommes obligés de brancher 4 fils au même point de connexion. Pourtant, dans la vraie vie on ne construit pas de systèmes où se branchent plusieurs fils au même point de connexion. Par exemple, une multiprise a plusieurs points de connexion, et elles sont toutes reliées à un même câble d'alimentation que l'on branche au courant. Un autre exemple est votre ordinateur: il possède plusieurs entrées USB, qui sont toutes reliées au même disque de mémoire qui va les lire.
La breadboard fonctionne donc de la même façon, avec des ensembles de points de connexion qui se regroupent en lignes ou colonnes s'ils sont reliés au même point de connexion.

Comprendre le fonctionnement d'une carte Arduino.

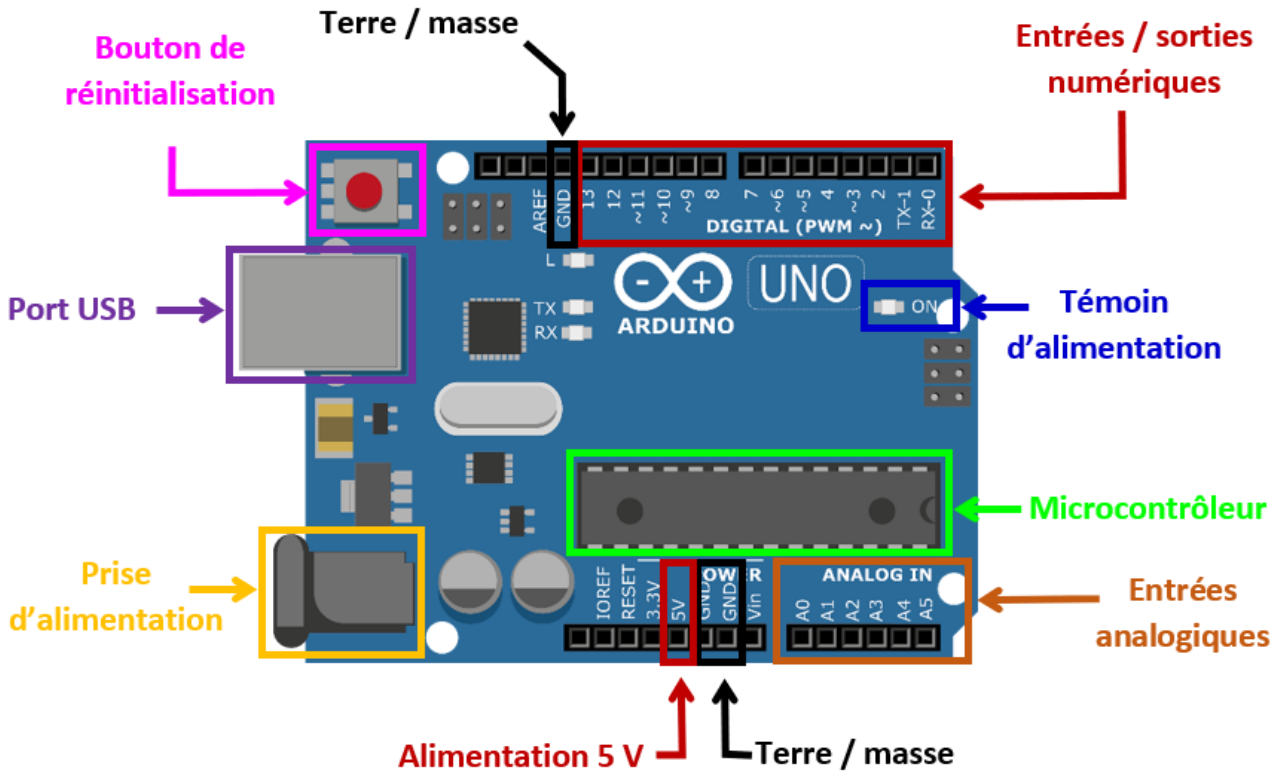


Fig. 8 : Schéma des éléments principaux d'une carte Arduino

Voici quelques éléments importants qui composent la carte Arduino (Fig. 8):

- Entrées/Sorties numériques : elles reçoivent et envoient de l'information sous forme numérique (entiers codés en 8 bits).
- Entrées analogiques : reçoivent des signaux continus dans le temps.
- Témoin d'alimentation : LED allumée quand l'Arduino fonctionne
- Microcontrôleur : « petit ordinateur » qui gère le code envoyé sur l'Arduino
- Alimentation (5V) : permet d'alimenter les composants
- Masse (0V/GND): pour relier les composants à la masse
- Prise d'alimentation : alimente la carte sans ordinateur
- Port USB : permet d'alimenter l'Arduino et de transférer un programme d'un ordinateur à l'Arduino

Connecter une carte Arduino et une breadboard.

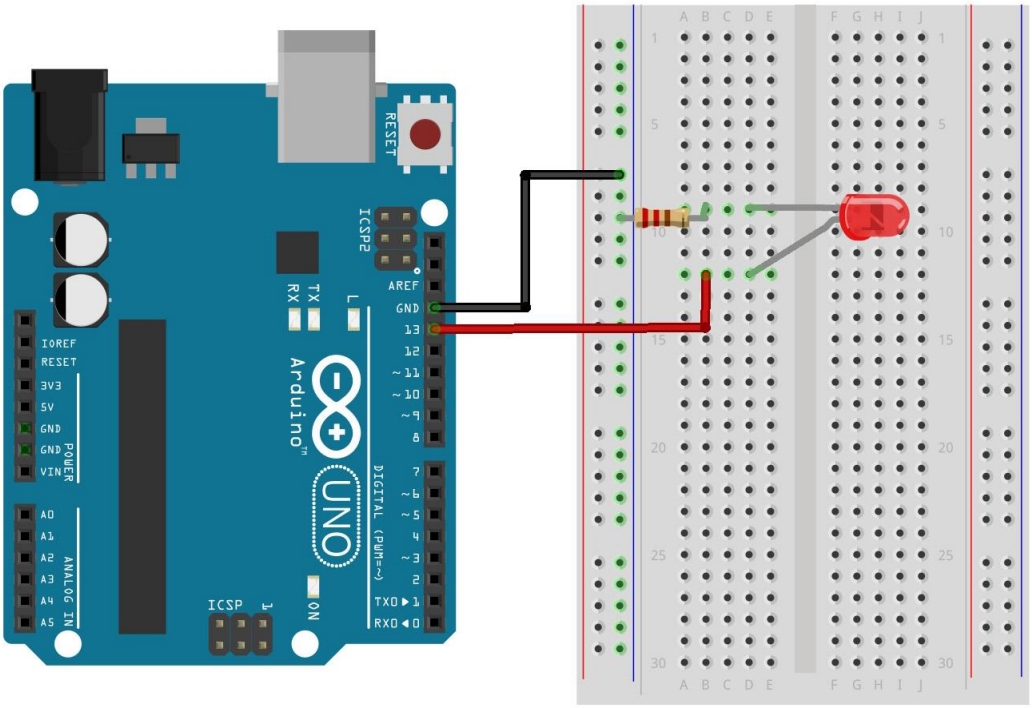


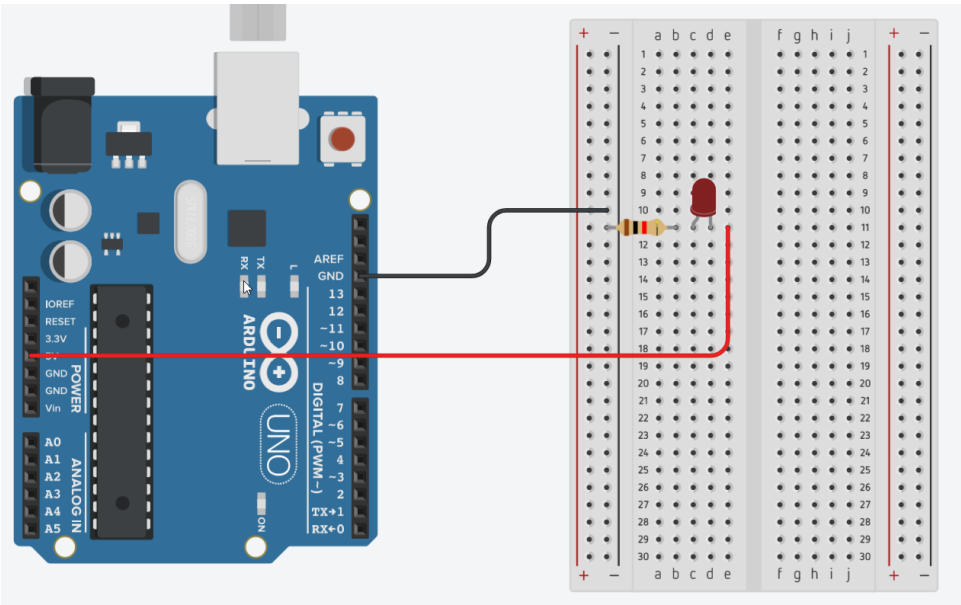
Fig. 9 : Schéma du montage en série LED-Résistance en Arduino

Voici un montage très simple d'une résistance en série avec une LED (Fig. 9).
Comme vous pouvez le voir, l'alimentation du circuit est fournie par le fil rouge, qui relie le circuit électrique à l'entrée 13 (une tension est ainsi appliquée).

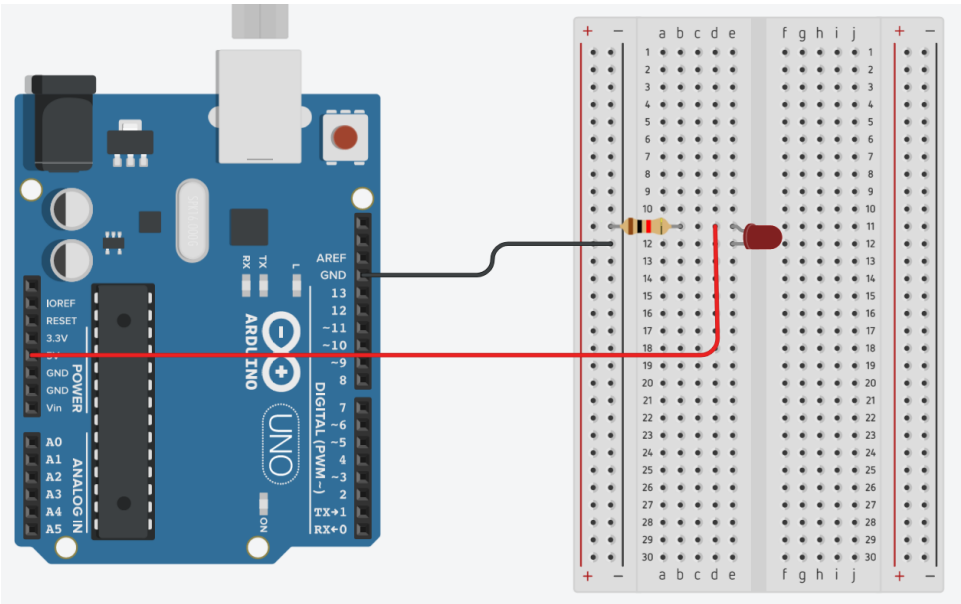
Remarquez que le fil rouge et la patte de la LED ne sont pas branchées au même trou : elles sont branchées sur deux trous différents sur une même ligne centrale horizontale. Ils sont donc reliés au même point de connexion. Finalement, le circuit se boucle en reliant la fin du circuit à la masse (fil noir branché au GND).

Est-ce-que tu as bien compris ?

1) Ce branchement est-il correct ?



2) Même question.



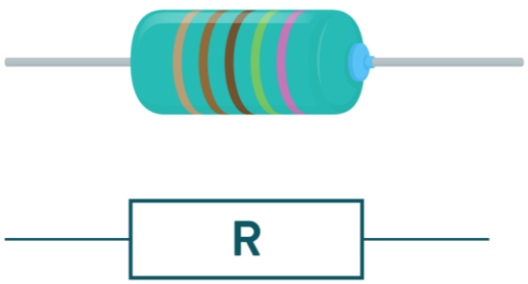
Étape 2 : LEDs et résistances

Objectifs:

- ❑ Comprendre l'intérêt des résistances;
- ❑ Allumer 1 LED sans programmation;
- ❑ Comprendre le code couleur des résistances;

Comprendre l'intérêt des résistances.

Schéma correspondant:



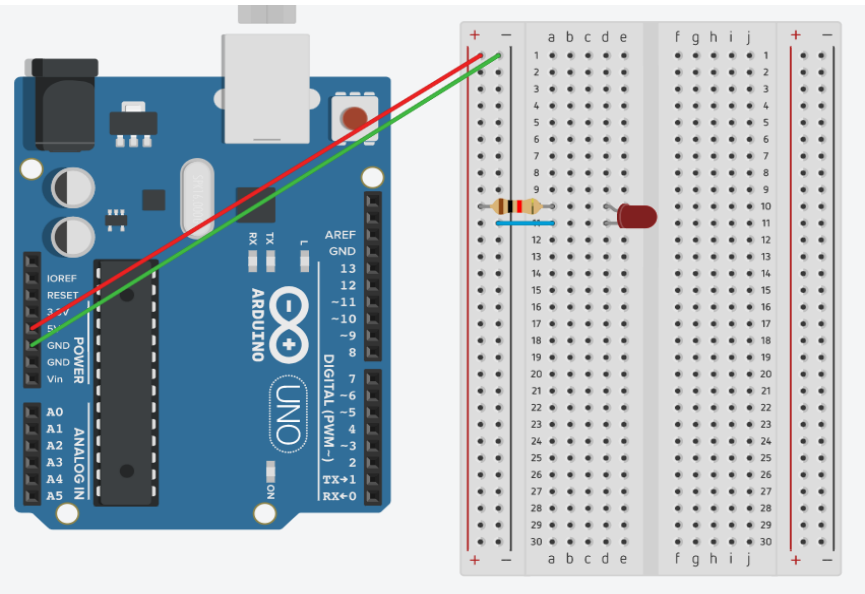
(a) Résistance

Les résistances ressemblent au dessin de la Figure (a), et sont schématisées par un rectangle. Leur rôle est simple, elles font diminuer le courant qui les traverse.

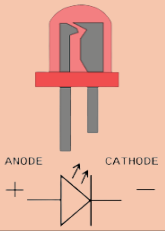
Mais pourquoi voudrait-on diminuer le courant passant dans un fil ? Car certains composants électriques grillent très facilement s'ils sont parcourus par un courant important. C'est le cas des LED !

Donc, si on veut brancher une LED, il faudra nécessairement brancher en série une résistance pour que la LED ne grille pas.

Allumer 1 LED sans programmation:



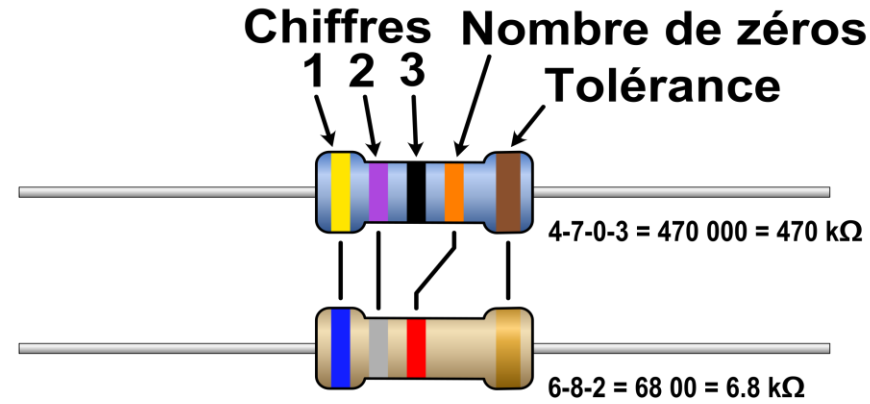
Remarque importante:
La **LED** a un **sens** (la patte longue correspond à celle qui est pliée sur le schéma).
Donc la LED s'allumera seulement si le courant est parcouru du + vers le -.



Fais le montage et vérifie avec l'encadrant !

Comprendre le code couleur des résistances:

Les résistances peuvent diminuer plus ou moins le courant. Cela dépend de leur valeur de résistance. Celle-ci se mesure en Ohm (dont le symbole est Ω). Plus la valeur de la résistance est grande, plus le courant va diminuer.
Les anneaux de couleur autour de la résistance ont un code couleur qui permet de savoir la valeur de la résistance. Ce code couleur est universel.



Chiffre	0	1	2	3	4	5	6	7	8	9

Tolérance	argent ±10%	or ±5%	±1 %	±0.5 %	±0.1 %

Est-ce-que tu as bien compris ?

1) Quelle est la valeur des résistances suivantes ?

R =

R =

R =

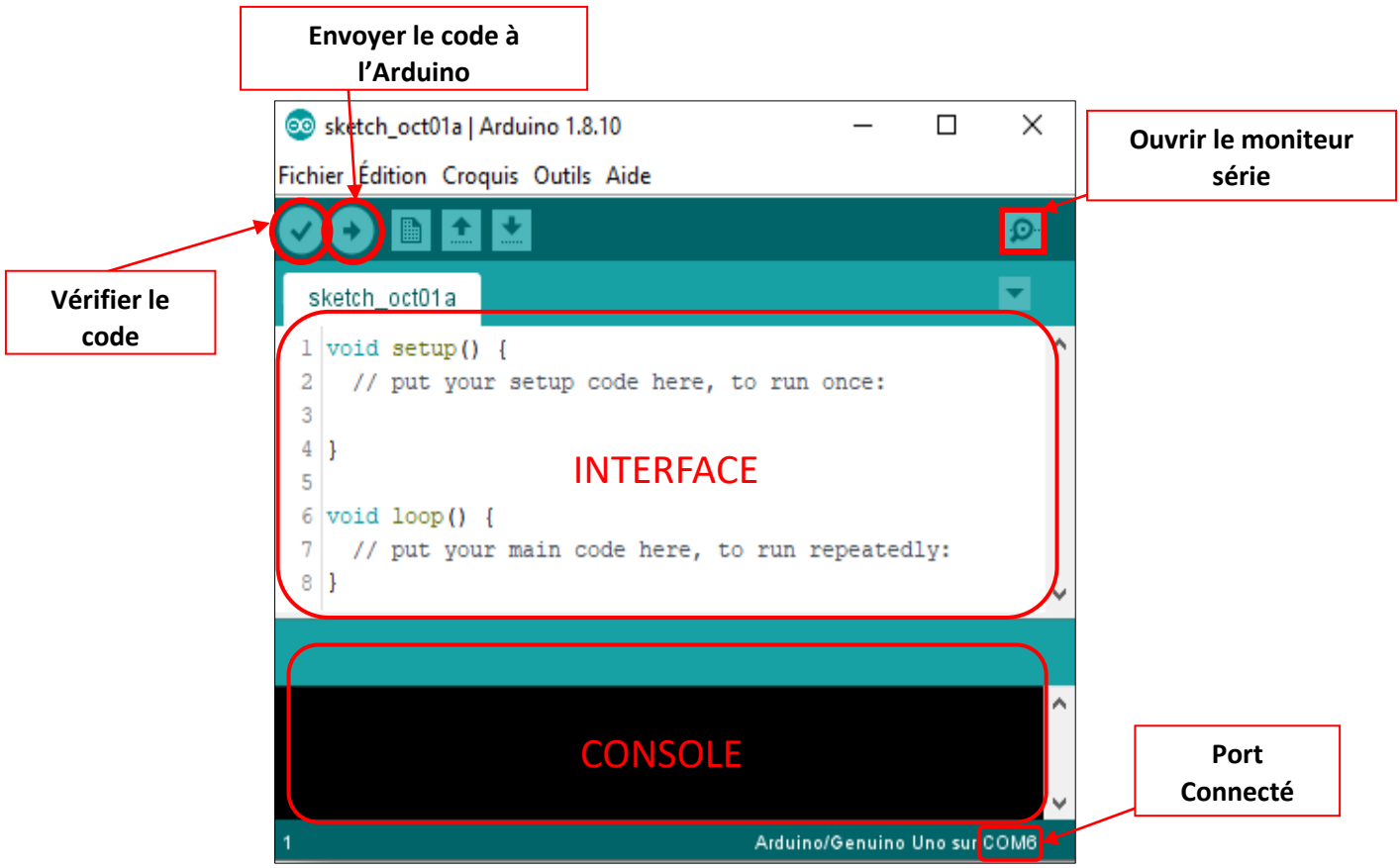
R =

Étape 3a : Programmation - Interface

Objectifs:

- ❑ Comprendre le fonctionnement de l'interface Arduino;

Comprendre le fonctionnement de l'interface Arduino.



Interface : L'interface est l'endroit où l'on écrit toutes les instructions (le code) que l'Arduino devra exécuter.

Console : La console est votre meilleur ami. C'est l'endroit où les erreurs apparaissent lorsqu'on demande de vérifier le code. Il indique quel type d'erreur et à quelle ligne elle s'est produite, ce qui facilite grandement la correction du code.

Moniteur Série : Il est possible que l'on demande au code d'afficher du texte ou des nombres. Ces valeurs s'afficheront dans le Moniteur Séries.

Étape 3b : Programmation - Variables

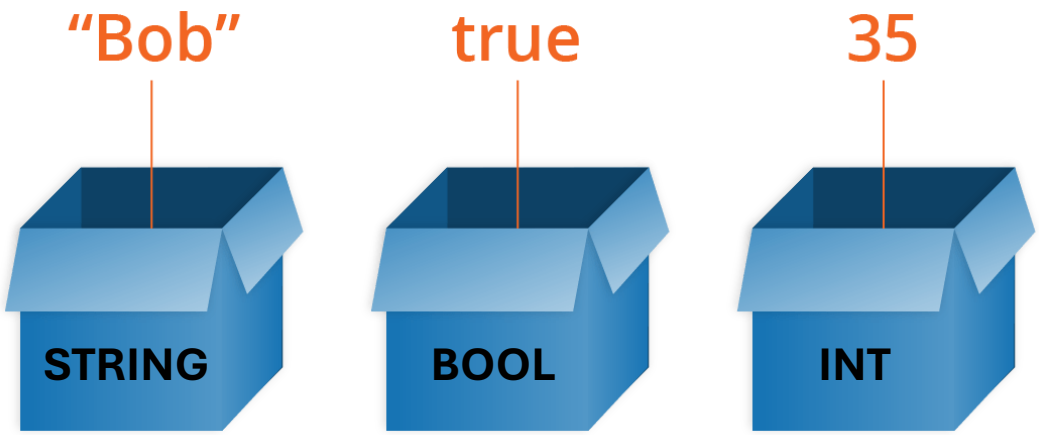
Objectifs:

- ❑ Comprendre le fonctionnement d'une variable;

Comprendre le fonctionnement d'une variable.

Une variable est une boîte dans laquelle on peut ranger des valeurs, textes, matrices, fonctions etc… Il en existe plusieurs types, voici celles que vous allez utiliser le plus souvent :

- **int** : Les variables int contiennent des nombres entiers. Ex : 0 , 1, 465, -34 etc…
- **float** : Les variables float contiennent des nombres à virgule. Ex : 0.0; 2.34; 435.65; 23.0; -32.3, etc…
- **string** : Les variables string contiennent des chaînes de caractères (des lettres, ponctuations, des chiffres etc…). Elles sont notées entre guillemets. Ex : "mot", "56" etc…
- **bool** : Les variables bool prennent deux valeurs : Vrai ou Faux
- **Tableau** : Les tableaux sont une liste de variables. Elles permettent de stocker plusieurs paramètres dans une seul variable. Ex : [6, "salut", true, 33, "bonjour"]



N'hésitez pas à utiliser l'ordinateur pour comprendre cette leçon

Afin d'utiliser une variable, il faut d'abord la déclarer (On dit au programme qu'il existe une boîte avec le nom de notre variable). Pour cela, au début de notre code, on écrit:

[Type de Variable] Nom_de_la_variable = Valeur_de_la_valeur_initiale;

Exemples

```
1. int C = 3;   // On crée une variable entière "C" à
   laquelle on associe la valeur 3
2. int F;       // On ne met rien dedans. La boîte est
   alors vide jusqu'à ce qu'on y mette quelque chose
3. float Nombre = 2.64;   // On crée une variable
   "Nombre" à laquelle on associe 2,64
4. bool verif = true;
```

? Pour mieux comprendre la déclaration d'une variable, déclarez une variable avec comme nom « Prénom » et comme valeur votre Prénom. Laquelle est correct entre « int N = "32"» et « int N = 32» ?

Après avoir déclaré la variable, on n'a plus besoin de rappeler son type lorsque l'on veut changer sa valeur, l'utiliser dans des opérations, l'utiliser dans des fonctions etc… Il suffit juste d'utiliser son nom.

Exemples

```
1. int C = 3;
2. F =3*C;
```

? Pour mieux comprendre l'opération de variables, déclarez deux variables que vous additionnerez ensuite dans une troisième variable.
? Que se passe-t-il quand on additionne 2 variable de type string?

Rappel: On peut vérifier la valeur de notre variable en utilisant la fonction Serial.println(Nom_Variable), puis en ouvrant le moniteur série.

Etape 3c : Programmation et variables

Objectif:

- ❑ Comprendre la structure d'un code Arduino
- ❑ Être capable d'utiliser les fonctions propres d'Arduino

La structure:

Arduino est un dérivé de la programmation en C+, elle est faite pour la robotique et donc possède une structure bien particulière. Elle se décompose en 2 parties principales et d'autres parties annexes.

Attention, une des particularités d'Arduino est qu'à chaque fin de ligne de code il faut mettre « ; ».
Il faut aussi faire attention au «{» lorsqu'une est ouverte, elle doit être refermée à un moment.

Les deux parties principales:

Le void setup:
C'est une partie du programme qui ne s'exécute qu'une seule fois

- Permet de définir nos variables en entré (INPUT), si c'est un capteur (exemple: le bouton) ou sinon en sortie (OUTPUT) (exemple: LED).
- Permet aussi de déclarer le moniteur série, qui permet de communiquer avec la carte.

Le void loop:
C'est ici que ce situe le corps du programme. Il tourne en boucle.

```
1
2
3
4   void setup() {
5     // put your setup code here, to run once:
6
7   }
8
9   void loop() {
10    // put your main code here, to run repeatedly
11
12  }
```

Parties secondaires:

Avant le void setup:
- On peut y définir les variables (voir fiches sur les variables).

Void fonction annexe:
C'est un sous-programme qui s'exécute une fois lorsque il est appelé (pour rendre l'ensemble plus lisible).

Les fonctions propres:
Arduino possède son propre langage, ce qui inclut une syntaxe particulière, et des fonctions de base très utile .

Dans le void setup:

pinMode():
Permet de dire si la broche est considérée comme une entrée ou une sortie.

Serial.begin(9600):
Permet de lancer le moniteur série.

EXAMPLE

```
1. const int LED = 5;     //Je dis que la LED est reliée à la broche 5
2.
3. void setup() {
4.   pinMode(LED, OUTPUT);    // Je déclare le LED en tant que sortie
5.   Serial.begin(9600);      // Je démarre le moniteur série
6. }
```

Dans le void loop:

digitalWrite:
Permet d'envoyer du courant dans une broche en sortie.
Même principe que précédemment pour 1 et 0.

EXAMPLE

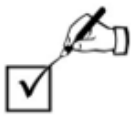
```
12   void loop() {
13   digitalWrite(LED,HIGH);        //allumer
14   digitalWrite(LED,LOW);         //éteindre
15   }
```

Les notations 0 ou LOW et 1 ou HIGH sont équivalentes, autrement dit pour la carte HIGH =1 et LOW =0

Delay(t):
Permet de mettre le programme en pause (le temps t est en ms)

Serial.println(...):
Permet d'écrire dans le moniteur série.

A toi de programmer! Tu peux essayer d'écrire sur le moniteur série « hello », ou essayer de faire clignoter une LED



Est-ce-que tu as bien compris ?

Expliquez précisément ce que fait ce programme.

```
1. const int LED_verte = 5;
2. const int LED_rouge = 6;
3.
4. void setup() {
5.   pinMode(LED_verte, OUTPUT);
6.   pinMode(LED_rouge, OUTPUT);
7.   Serial.begin(9600);
8. }
9.
10.void loop() {
11.  digitalWrite(LED_verte,HIGH);
12.  Serial.println("la LED verte est allumée")
13.  delay(1000)
14.  digitalWrite(LED_verte,LOW);
15.  digitalWrite(LED_rouge,HIGH);
16.  Serial.println("la LED verte est éteinte et
   la LED rouge est allumée");
17.  delay(1000);
18.}
```

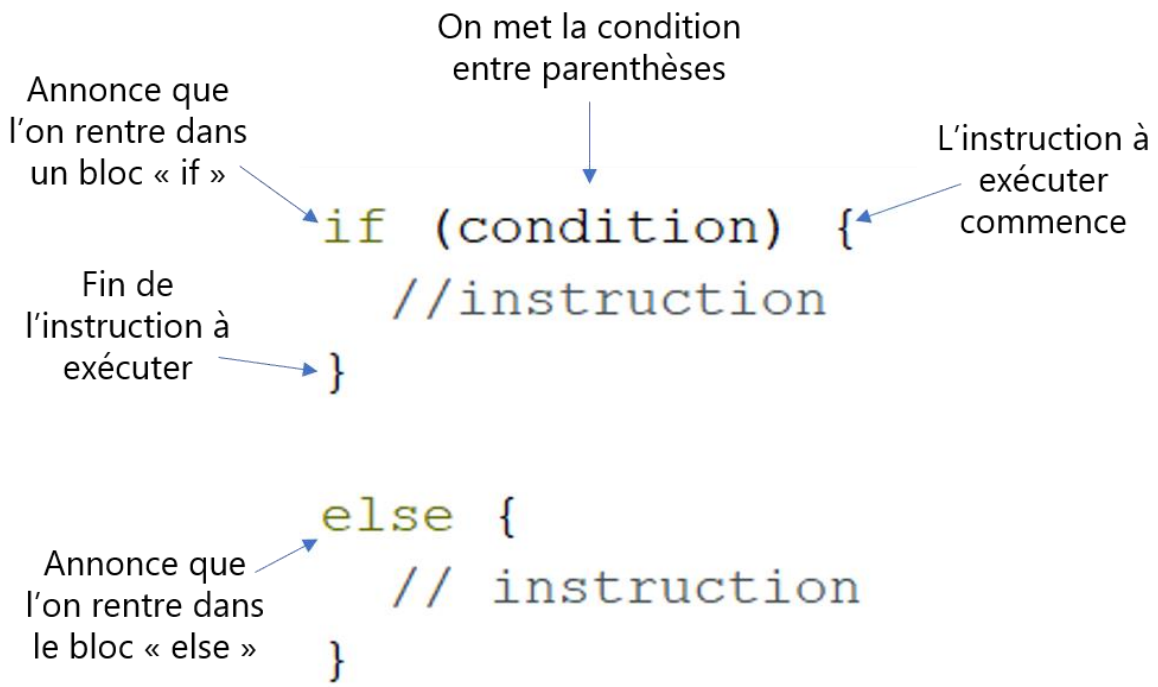
Étape 3d : Programmation - Blocs

Objectifs:

- ❑ Comprendre le fonctionnement d'un bloc **<< if/else >>;**
- ❑ Comprendre le fontionnement d'un bloc << **while >>;**
- ❑ Comprendre le fonctionemment d'un bloc << **for >>;**
- ❑ Comprendre les conditions et opérations logiques.

Comprendre le fonctionnement d'un bloc **<< if/else >>**

Un bloc conditionnel est un moyen d'exécuter des instructions seulement si une condition est remplie. Si la condition est vérifiée, l'instruction du bloc « if » est exécuté. Sinon, c'est celle du bloc « else ». Voilà comment faire :



Exemple simple:

```
if (il fait nuit){
  J'allume la lumière
}
else {
  J'éteins la lumière
}
```

Les **conditions** sont normalement liées aux états des **entrées**, telles que les boutons. Les **instructions** sont associées aux **sorties**, comme allumer des LED ou produire des sons avec un buzzer.

Comprendre le fonctionnement d'un bloc **<< while >>**

Un bloc "while" est également une structure conditionnelle, mais sa structure diffère légèrement. Il fonctionne comme une boucle, où les instructions à l'intérieur se répètent tant que la condition est vraie. Lorsque la condition n'est plus satisfaite, il sort du bloc "while" et poursuit l'exécution du code. Voilà comment faire :

```
while (condition) {
  // instruction
}
```

? Pour mieux comprendre le fonctionnement de la boucle while, remplace le bloc « if » ci-dessus par une boucle « while » pour allumer la lumière tant qu'il fait nuit.

Attention: pour écrire des conditions, on a besoin de comprendre les possibles tests qu'on peut faire avec nos variables.

Conditions

Pour les conditions, nous avons plusieurs options :

1- Égalité : Pour vérifier si une variable est égale à une valeur, nous utilisons l'opérateur "==" (variable == valeur). Fais attention à utiliser deux signes égaux pour l'opération d'égalité.
2 - Inégalité : Pour vérifier si une variable est différente d'une valeur, nous utilisons l'opérateur "!=" (variable != valeur).
3 - Comparaison : Pour effectuer des comparaisons, nous utilisons les opérateurs de comparaison tels que "<=" (inférieur ou égal à) et ">=" (supérieur ou égal à) (variable <= valeur et variable >= valeur).

```
1. int A = 3;
2. if (A==3) {
3.   Serial.println("A vaut 3")
4.   }
```

Exemple simple

Opérations Logiques

Il est également possible d'avoir plus d'une condition et d'utiliser des symboles pour identifier les opérations "ET" et "OU" :

1 - "ET" logique : Pour vérifier si deux conditions sont toutes les deux vraies, nous utilisons l'opérateur "&&" (par exemple : (condition1) && (condition2)). Cette condition sera vraie uniquement si les deux conditions sont satisfaites.
2 - "OU" logique : Pour vérifier si au moins l'une des deux conditions est vraie, nous utilisons l'opérateur "||" (par exemple : condition1 || condition2). Cette condition sera vraie si l'une des deux conditions est satisfaite.

```
1. int A = 3;
2. string B = "salut"
3. if (A==3 && B=="salut") {
4.   Serial.println("A vaut 3 et B vaut 'salut'")
5.   }
```

Exemple simple

Comprendre le fonctionnement d'un bloc << **for** >>

Le bloc "for" est une structure de programmation qui permet de répéter un ensemble d'instructions un certain nombre de fois.
Pour mieux comprendre: Imagine que tu as une boîte de bonbons et que tu veux en manger un chaque jour pendant une semaine. Tu pourrais utiliser une boucle "for" pour t'aider. Voici un exemple simple de boucle "for" pour manger un bonbon chaque jour pendant une semaine :

```
for (int jour = 1; jour <= 7; jour++) {
  mangerBonbon();
}
```

Dans cet exemple, le bloc « for » commence en créant une variable appelée "jour" et l'initialise à 1. Ensuite, il spécifie que tant que "jour" est inférieur ou égal à 7, les instructions à l'intérieur du bloc "for" sont répétées. Finalement, "jour++" signifie que, à chaque tour de boucle, on incrémente de 1 la valeur de jour. À chaque itération de la boucle, nous appelons la fonction "mangerBonbon()" pour manger un bonbon. La boucle continue jusqu'à ce que "jour" atteigne la valeur 8, puis elle se termine.

C'est un moyen pratique d'effectuer une action répétée un nombre spécifique de fois. Voici la structure en résumé:

```
for (initialization; condition; increment) {
  //instruction
}
```

Est-ce-que tu as bien compris ?

1) Expliquez pourquoi les codes ne sont pas corrects.

```
for (i=1; i++; i<4) {
  //instruction
}
```

```
if(state = 1){
  //instruction
}
```

```
while (state != 1):
  //instruction
```

2) Pourquoi dans la boucle **for** on écrit « variable = valeur » et dans le bloc **if** on écrit « variable == valeur » ?

Étape 4a : La notion de fréquence

Objectifs:

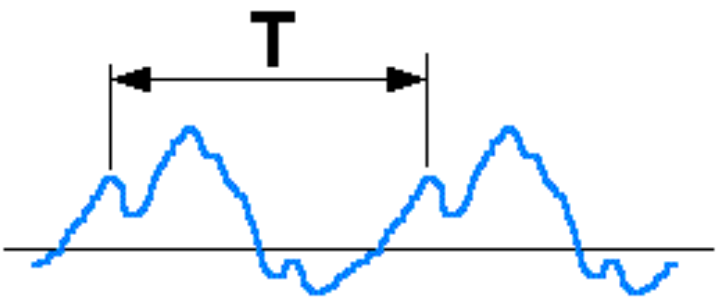
- ❑ Maîtriser la notion de fréquence

Rappel : Notion de fréquence

On a parlé en page précédente de « fréquence » d'allumage. Il faut bien garder en tête, notamment pour ce qui suit que, dans notre cas, **plusieurs fréquences sont mises en jeu**.

Pour bien le comprendre, rappelons la définition propre de fréquence. Par définition, la fréquence (notée f) d'un **phénomène périodique** correspond au nombre de répétitions de ce dernier pendant une seconde. La fréquence est donc associée à un **signal périodique**.

La notion de fréquence est étroitement liée à celle de **période.** En effet, on a $f = \frac{1}{T}$. Plus qualitativement, la période d'un signal périodique correspond à l'intervalle de temps entre deux répétitions successives d'un instant du signal.



Étape 4b : Clignotement d'une LED

Objectifs:

- ❑ Faire clignoter une LED
- ❑ Choisir une fréquence de clignotement

Faire clignoter une LED

Pour faire clignoter une LED, c'est très simple : on va naturellement l'éteindre (état LOW) puis l'allumer, (état HIGH) en oubliant pas d'attendre entre les deux. Attention, la fonction delay prend un temps en ms

? Exercice pratique : écrire les 4 lignes à insérer dans le `void loop` pour faire clignoter la LED. On attendra **500ms** entre l'allumage et l'extinction.

```
1.  const int LED = 11;
2.
3.  void setup() {
4.    pinMode(LED, OUTPUT);
5.    Serial.begin(9600);
6.   }
7.   void loop() {
8.     ……
9.  }
```

…….
…….
…….
…….

? Question pratique : Si on s'intéresse au signal correspondant à l'**état** (allumée/éteinte) de la LED. Quelle est **la fréquence** de ce signal ?

…….
…….

On souhaite désormais faire clignoter la LED à une fréquence $f = 1\ Hz$.

? Exercice pratique: compléter l'argument de la fonction delay. Tester le code. Commenter.

```
1. const int LED = 11;
2.
3. void setup() {
4.    pinMode(LED, OUTPUT);
5.    Serial.begin(9600);
6.   }
7.   void loop() {
8.    digitalWrite(LED,HIGH);
9.    delay(……);
10.   digitalWrite(LED,LOW);
11.   delay(……);
12.}
```

……
……
……
……
……
……
……
……
……

Étape 5: bouton

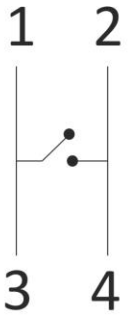
Objectifs:

- ❑ Comprendre le fonctionemment d'un bouton;
- ❑ Allumer 1 LED avec un bouton, sans programmation;
- ❑ Allumer 1 LED avec un bouton, avec programmation;
- ❑ Allumer 4 LEDs avec quatre boutons correspondents;

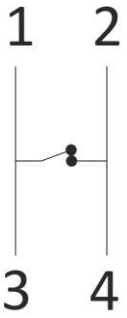
Comprendre le fonctionemment d'un bouton.

Schéma correspondant:

(a) Bouton



(b) Bouton relâché



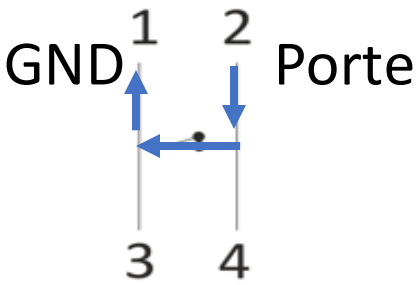
(c) Bouton pressé

Le bouton du kit Arduino ressemble à celui de la Figure (a). Pour utiliser le bouton correctement, nous allons connecter un fil à la patte 1 du bouton et un autre fil à la patte 2. Ainsi, le courant pourra circuler d'un fil à l'autre.

Dans ce cas, les points 1 et 3 ont le même potentiel, tout comme les points 2 et 4, ce qui signifie que nous n'avons pas besoin d'utiliser les pattes 3 et 4.

Lorsque le bouton est relâché, le circuit est ouvert, comme indiqué sur la Figure (b), et aucun courant ne circule dans les fils.

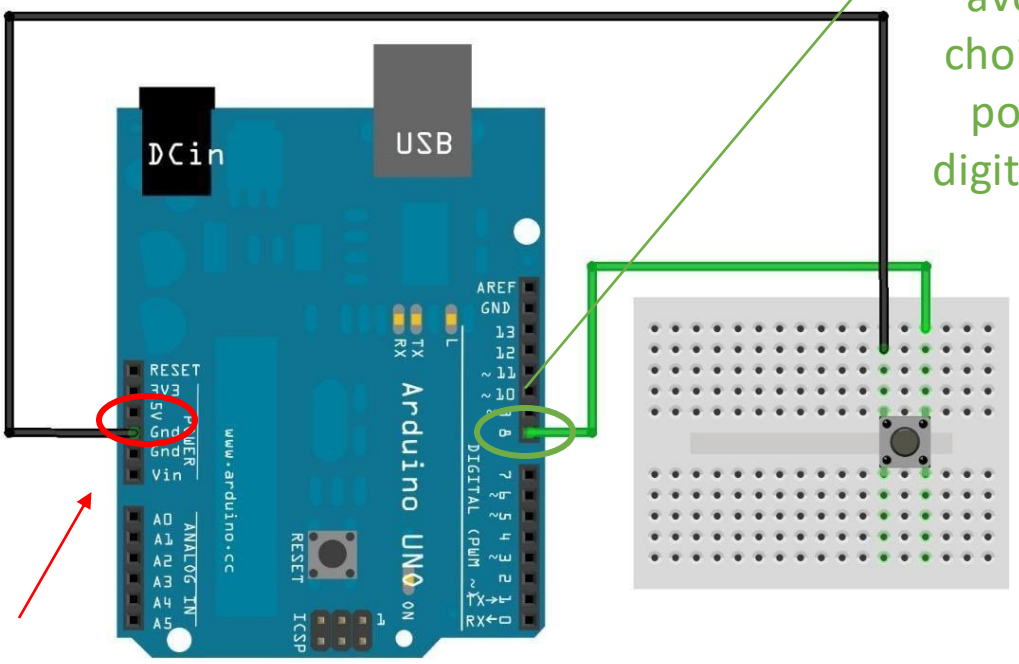
Après avoir appuyé sur le bouton, Figure (c), le courant circule dans le sens 2-1 si nous avons connecté le +5V à la patte 2 et la masse (GND) à la patte 1. Avec l'Arduino nous ne ferons pas la connexion directe avec +5V, nous allons faire une connexion avec une porte spécifique.



Chemin de la currant.

Il est possible de réaliser d'autres connexions, telles que 1 et 4, ou encore mettre la porte en 1 et GND en 2 (dans ce cas, le courant circulera dans le sens 2-1);

Montage avec Arduino:

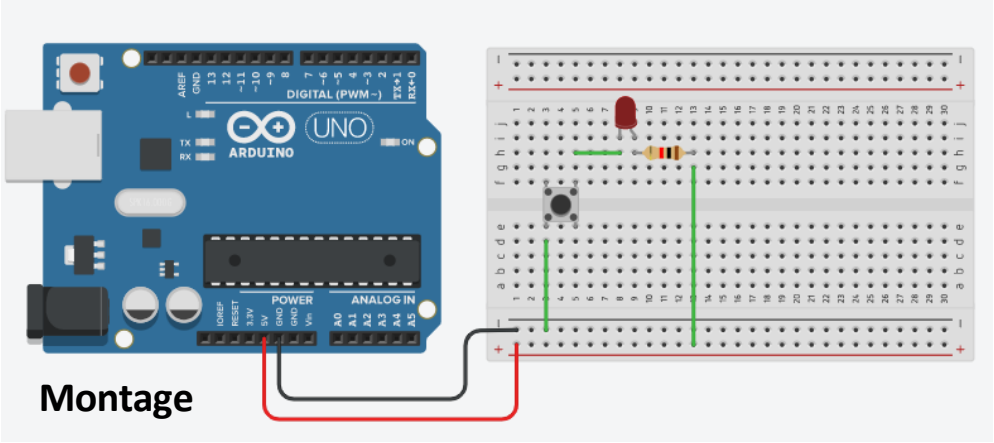


Et ici nous avons choisi la porte digital 9 !

GND c'est la masse de notre circuit !

Fait le montage et vérifie avec l'encadrant !

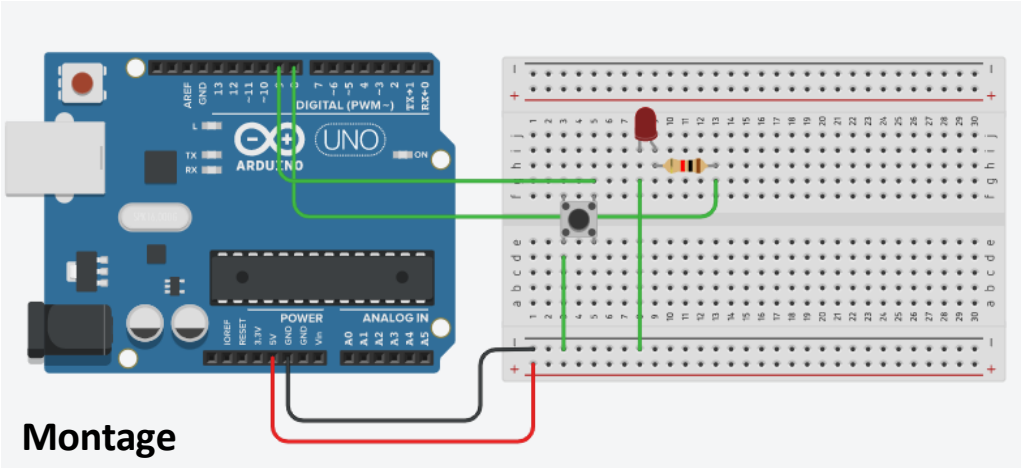
Allumer 1 LED avec un bouton, sans programmation:



? Pour mieux comprendre le fonctionnement de ce circuit, décrivez le parcour du courant. Avec des fléches, comme avant.
? La LED s'allume avec le bouton presse ou relâché ?

Fait le montage et vérifie avec l'encadrant !

Allumer 1 LED avec un bouton, avec programmation:



Fait le montage et vérifie avec l'encadrant !

Programmation: Pour la programmation on a besoin d'écrire avant la fonction setup la définition des broches de la LED et du bouton. Dans la fonction setup(), il faut définir le mode de la broche de la LED et le mode de la broche du bouton. Et dans la fonction loop(), il faut vérifier si le bouton est pressé et alors allumer la LED;

```
4  int led = 8;
5  int button = 9;
6  int button_state = 0;
7
8    void setup()
9  {
10   pinMode(led, OUTPUT);
11   pinMode(button, INPUT_PULLUP);
12 }
13
14 void loop()
15 {
16   button_state = digitalRead(button);
17   if(button_state == ?){
18     digitalWrite(led,HIGH);
19   }else{
20     digitalWrite(led,LOW);
21   }
22 }
```

Définition des ports et des variables de contrôle.

Définition des modes pour la LED et le bouton.

Vérification de l'état du bouton et commande de la led.

Mode: Nous allons utiliser le mode INPUT_PULLUP pour configurer la broche d'entrée du bouton. Ce mode permet de maintenir un état haut lorsque le bouton est relâché et un état bas lorsque le bouton est pressé.

digitalRead(): Cette fonction permet de lire l'état de la broche d'entrée du bouton. Elle retourne soit HIGH (état haut) lorsque le bouton est relâché, soit LOW (état bas) lorsque le bouton est pressé.

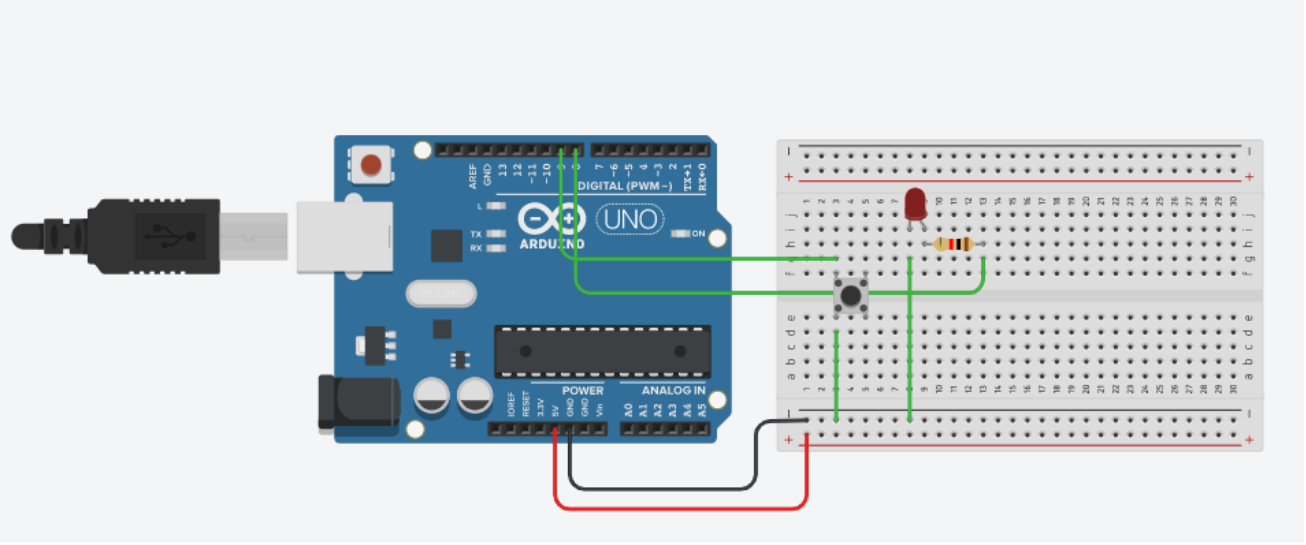
? Si l'on souhaite allumer la LED lorsque le bouton est pressé, quelle entre ces deux vérifications doit-on effectuer dans la ligne 17 ?

- ❑ if(bouton_state == LOW)
- ❑ if(bouton_state == HIGH)

Fait la programmation et vérifie avec l'encadrant !

Est-ce-que tu as bien compris ?

1) Est-ce que cette montage est bonne ? Expliquez le comportment du LED si on utilise le même code d'avant.



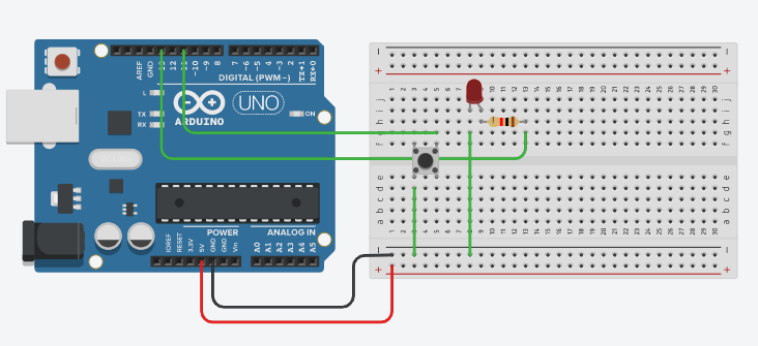
_____

_____

_____

_____

2) Completez le code en regardant la montage ci dessous:



```
int led = ____;
int button = ____;
int button_state = 0;

  void setup()
{
  pinMode(led, _______);
  pinMode(button, __________);
}

void loop()
{
  button_state = digitalRead(__________);
  if(button_state == _______){
    digitalWrite(led,_____);
  }else{
    digitalWrite(led,_______);
  }

}
```

Projet final

Objectifs:

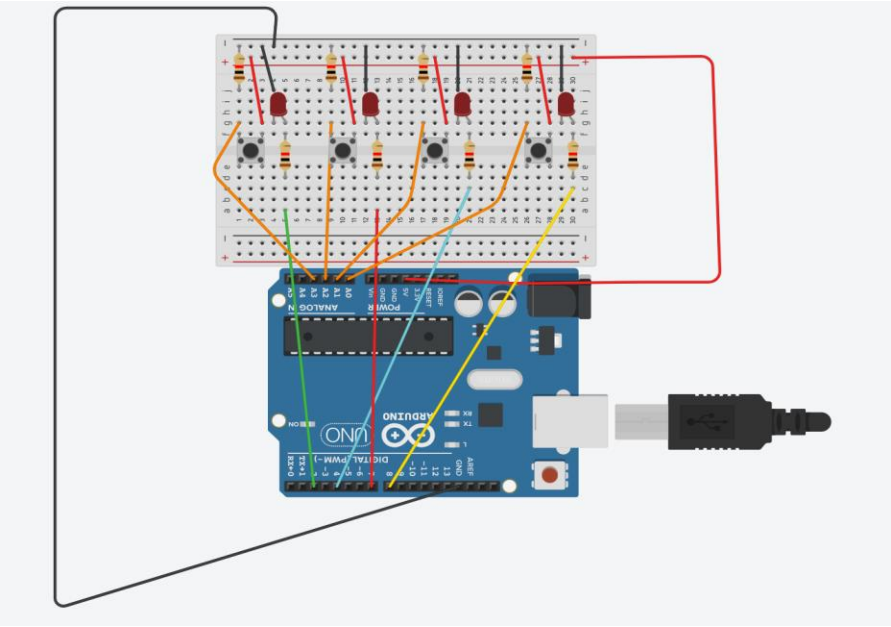
- ❑ Réaliser le montage électrique du jeu
- ❑ Compléter le corps du programme
- ❑ Améliorer le jeu (pour les plus rapide)

Règle du jeu:

Le but du jeu est d'aller le plus loin possible dans les niveaux. Pour chaque niveau, une séquence de lumières s'allume de façon aléatoire. Le joueur doit mémoriser la séquence et appuyer sur les boutons correspondant pour retrouver la séquence.
Quand un niveau est réussi, on passe au niveau suivant en ajoutant une couleur aléatoire à la séquence. Lorsqu'un joueur perd, on veut savoir son score, et remettre le jeu à 0.

Etape 1: Le Montage

A l'aide des fiches précédentes, de vos connaissance et du schéma ci-dessous, réalisez le montage électrique.



Fait le montage et vérifie avec l'encadrant !

Etape 2: Le Code

A l'aide de la structure de code disponible, des fiches précédentes, et de vos connaissance, complétez le code.
La structure du code donné, est la structure du jeu le plus simple possible, nous verrons dans les prochaines étapes que nous pouvons l'améliorer.

Pour compléter le code, il faut bien lire les conseiller situé en haut du code. De plus à chaque endroit où il faut compléter, des indications sur ce qui est attendu est disponible.
Il est aussi conseillé dès qu'il y a des difficultés, de relire les fiches précédentes.
Chaque ligne de code est écrite dans un code simple et pas astucieux. Il ne faut donc pas aller chercher les solutions trop loin.

Fait la programmation et lance le programme!

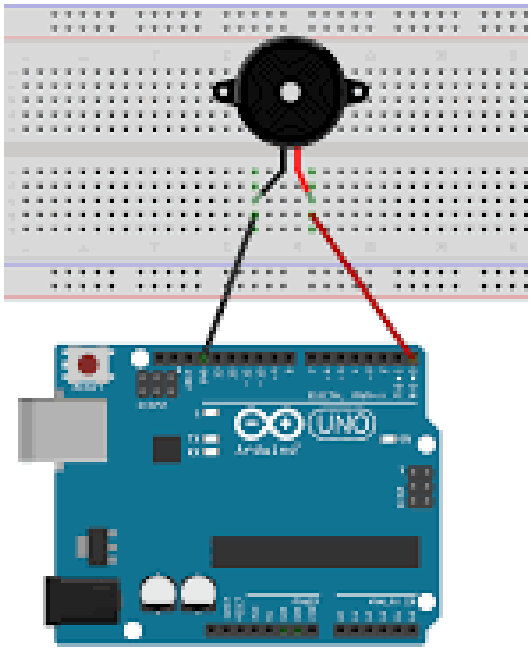
Etape 3: Améliorations (Pour les plus rapides)

Plusieurs améliorations simples et amusante sont possible. Vous pouvez laisser libre cours à votre imagination, ou suivre nos suggestions dans l'ordre que vous souhaitez.

1er suggestion: Le Buzzer (niveau moyen)

Un buzzer est un petit hautparleur capable de donner plusieurs notes avec une intensité réglable.

Pour les branchement il suffit de suivre le schéma. Cependant attention à mettre le buzzer sur une sortie analogique ~ par exemple la sortie 11 convient très bien.



Fait le montage et vérifie avec l'encadrant !

Pour le code voici un exemple de code, à vous de le déchiffrer et de l'utiliser dans votre jeu.
Notons tout de même que « tone » permet de définir la note à laquelle est joué le note et on utilise un analogueWrite car la sortie que l'on donne est un entier entre 0 et 1024 qui défini la puissance du sons émis.

```
1   int buzzer = 11;
2
3   void setup() {
4   pinMode(buzzer;OUTPUT);
5
6   }
7
8   void loop() {
9
10  tone(buzzer;500,500);      // allumer le buzzer au tone 500 ( les tones vont de 0 à 1024), pendant 500ms
11  analogueWrite(buzzer,0);   // éteindre le buzzer
12  delay(500);
13  }
```

Fait la programmation et lance le programme!

2ème amélioration: le potentiomètre (niveau facile)

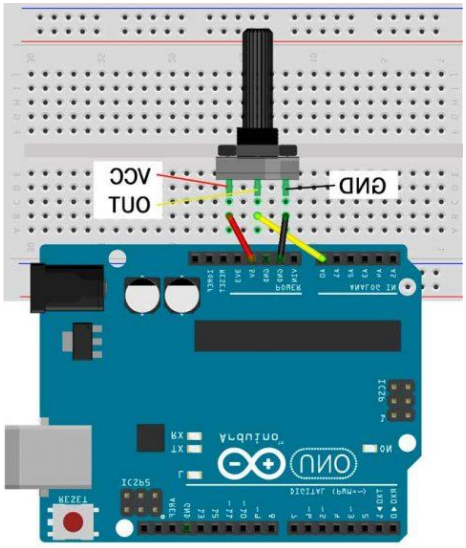
Il faut ici ajouter un potentiomètre qui permet de gérer la vitesse d'apparition du code couleur.

? Point de cours:

Pour commencer, un potentiomètre est une résistance réglable. Autrement dit, on peut réglé le R de la résistance en tournant. Cela permet, en utilisant la loi d'Ohm U=RI, d'obtenir une sortie analogique qui va de 0 à 1024. En effet comme on connait U, ici 5V, alors en fonction de comment on règle notre potentiomètre (R), alors on obtient un I. Le potentiomètre relève ce I et grâce à une amplification donne à la carte une valeur de 0 à 1024.

Comment fait-on:

Voici le schéma de branchement du potentiomètre, on récupère l'information analogique via la broche du centre.



Fait le montage et vérifie avec l'encadrant !

Voici un exemple de code pour comprendre comment il fonctionne.

```
1   int pot = A0;
2
3   void setup() {
4   pinMode(pot,INPUT);
5   Serial.begin(9600);
6   }
7
8   void loop() {
9   Serial.print(analogRead(pot));    // permet ici d'afficher sur l'ordinateur (moniteur série) la valeur
10                                    // du potentiomètre
11
12  }
```

Finalement l'idée est de récupérer l'information du potentiomètre et de l'adapter pour qu'il corresponde à un temps de delay entre l'allumage des LED. Autrement dit il faut remplacer les delay de 500 ou 1000 par une variable directement lié au potentiomètre.