DÉBUTER AVEC ARDUINO

SOMMAIRE

ARDUINO ... C'EST QUOI ET ÇA SERT À QUOI ???

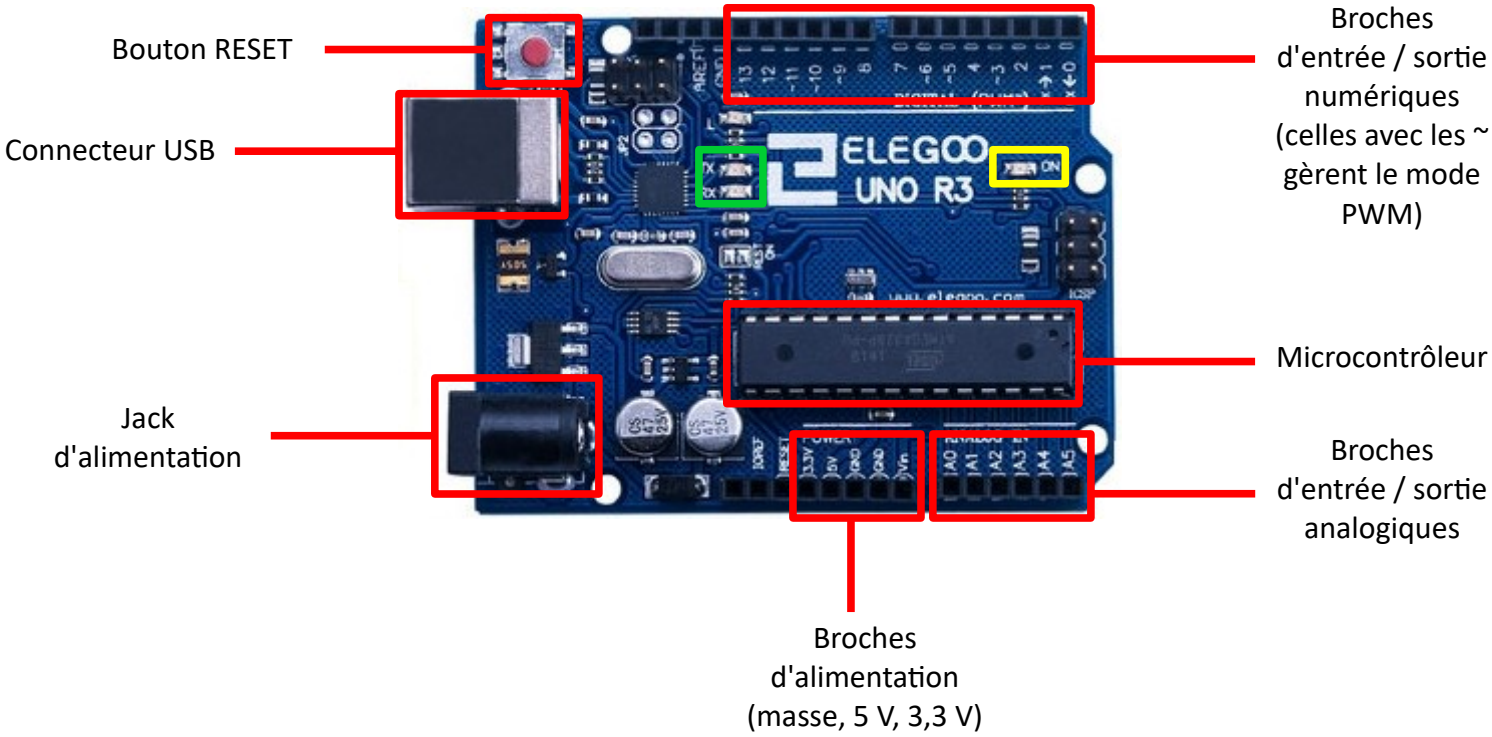
ARDUINO est un circuit imprimé comportant, entre autre, un microcontrôleur, des composants permettant de le faire fonctionner (alimentation, etc ...), de la connectique destinée à le faire communiquer avec un ordinateur, de l'inclure dans un circuit, ou encore d'étendre ses possibilités.

Le microcontrôleur d'ARDUINO est un circuit intégré rassemblant un microprocesseur, de la mémoire et des périphériques d'entrée - sortie. ARDUINO comporte par exemple des **C**onvertisseurs **A**nalogique **N**umérique et un générateur de signaux à modulation de largeur d'impulsion (en langage pas de chez nous, **P**ulse **W**idth **M**odulation) dont l'utilité sera développée plus tard.

L'intérêt des microcontrôleur est qu'ils sont programmables et qu'une fois cela fait, ils sont autonomes mais bien sûr, en étant alimentés. Ils pourront donc être utilisés dans des domaines d'application très variés qui peuvent aller de la gestion de l'arrosage d'un jardin au serveur Web en passant par la station météo personnelle sans oublier la robotique.

Il est à noter que le projet ARDUINO à été développée dans une école de design italienne dans les années 2000, la carte UNO R3 que l'on utilisera a été lancé en 2010. Le langage utilisé pour la programmation d'ARDUINO est développée en java et est très proche du langage C et du C++. Ce langage est opensource, multiplateformes et les schémas des cartes ARDUINO circulent librement. Il existe donc un ensemble de cartes clones (comme celle présentée ci-dessous) développées en parallèle des cartes ARDUINO officielles.

SE REPÉRER SUR LA CARTE ARDUINO

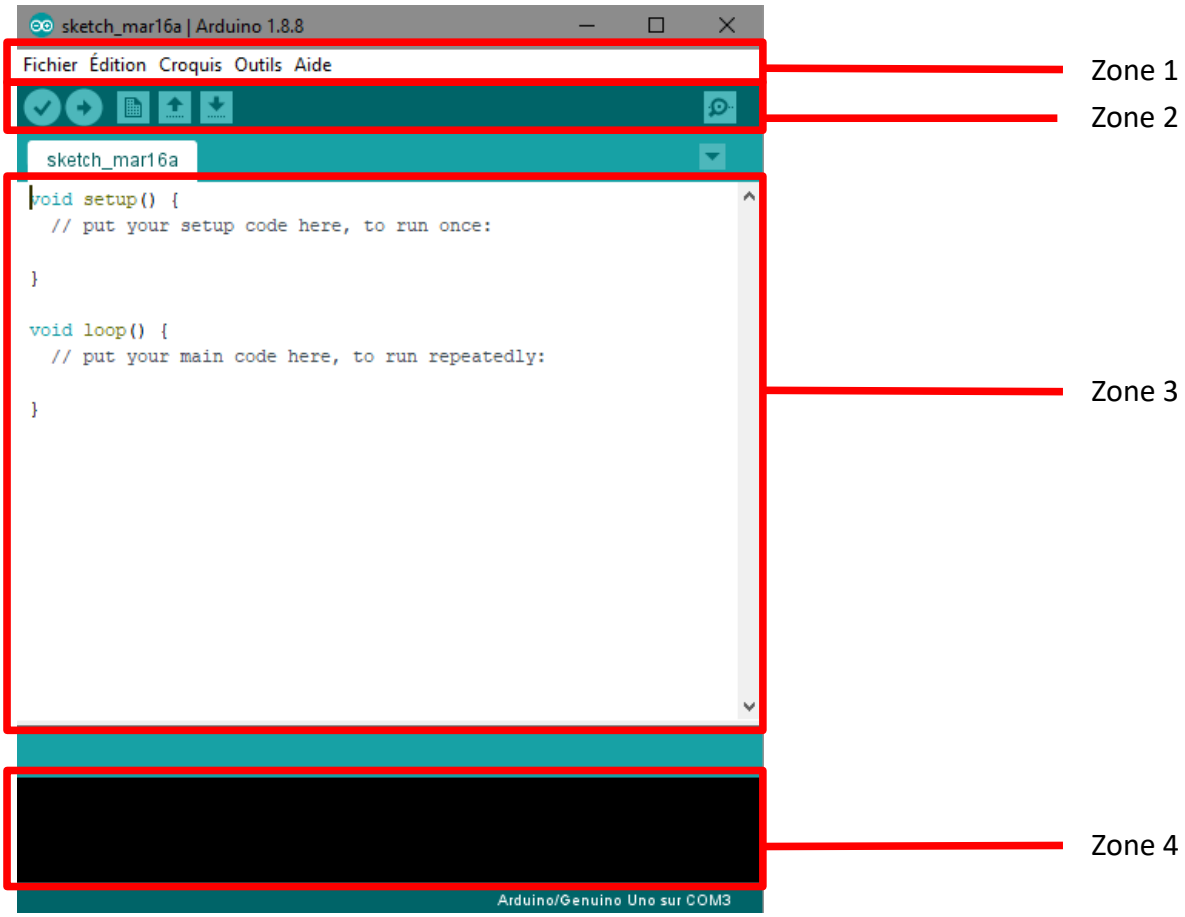


On trouve aussi une LED témoin d'alimentation (LED 1 entourée en jaune) et deux LED montrant l'activité du port série (LED TX et RX entourées en vert). Ces LED fonctionnent par exemple quand un programme est envoyé depuis l'ordinateur vers la carte ARDUINO (on dit que le programme est **téléversé**).

INTERFACE DU LOGICIEL

Au préalable, il est nécessaire de télécharger l'ensemble des éléments permettant l'utilisation de la carte à l'adresse **https://www.arduino.cc/en/Main/Software** puis ensuite de procéder à leur installation ou encore installer les éléments fournis par le fabricant de la carte.

Les cartes se programment grâce à un environnement de développement intégré (en langage pas de chez nous, **I**ntegrated **D**evelopment **E**nvironment) dont la fenêtre est la suivante :



→ La zone 1 est la barre de menu du logiciel. On y trouve, dans le menu **Fichier**, les commandes classiques pour enregistrer / ouvrir des fichiers (dans la terminologie propre à ARDUINO, ceux-ci sont appelés croquis ou sketch en anglais), les imprimer, etc ...

On va trouver dans le menu **Outils**, les commandes permettant de préciser quelle type de carte est utilisé, sur quel port elle est connectée, une commande de gestion des bibliothèques, une commande permettant de démarrer le moniteur série. Ces commandes seront détaillées ultérieurement.

→ La zone 2 est une barre d'outils. Dans l'ordre (en partant de la gauche), les boutons qui la composent permettent :

- de vérifier (en le compilant) le programme réalisé.
- de téléverser le programme réalisé (rappel : le téléversement est l'envoi du programme vers la carte).
- de créer un nouveau fichier (rappel : un fichier est appelé croquis).
- d'ouvrir un croquis
- d'enregistrer un croquis
- de démarrer le moniteur série

→ La zone 3 est la partie de la fenêtre dans laquelle la saisie du programme s'effectue (les mot-clés utilisés par le langage ARDUINO apparaissent en orange).

→ La zone 4 est la console permettant l'affichage des résultats de compilation, de téléversement, etc ...

ON COMMENCE EN TESTANT LE MATÉRIEL ...

Avant de démarrer, il est nécessaire de faire en sorte que la carte Arduino soit reconnue par l'ordinateur :

- Connecter la carte Arduino à un port USB de l'ordinateur.
- Démarrer l'IDE.
- Dans le menu **Outils**, cliquer sur **Type de carte** et choisir **Arduino / Genuino Uno**.

  Toujours dans le menu **Outils**, cliquer sur **Port** et choisir le port sur lequel il est noté **(Arduino / Genuino Uno)**.

La carte Arduino est maintenant reconnue par l'ordinateur et prête à être programmée :

- Dans la zone 3 de la fenêtre de l'IDE, saisir le texte suivant en respectant les majuscules / minuscules :

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

- Téléverser le programme ; l'IDE indique dans la console (zone 4) les différentes étapes : compilation puis téléversement. Un message s'affiche lorsque celui-ci est terminé.
- Observer la carte Arduino : une LED orange située à côté des diodes TX et RX devrait clignoter.

La carte fonctionne et vous venez de réaliser votre premier programme et votre premier téléversement.

ANALYSE DU PROGRAMME SAISI : COMMANDES D'ENTRÉE / SORTIE

Un programme comporte au minimum deux procédures débutant par la commande **void()**, la procédure nommée **setup()** et la procédure **loop()**. On remarquera que le bloc d'instructions exécutées par chacune des procédures est délimité par deux accolades et que chaque instruction est suivie du signe ;.

La première procédure - setup() - est la procédure qui contiendra les différentes commandes de configuration de la carte (définition des broches utilisées en entrée ou en sortie, débit de communication, etc ...). La seconde - loop() - est la procédure qui s'exécutera en boucle jusqu'à ce que la carte soit mise hors tension ou que le bouton RESET soit pressé.

La procédure setup() du programme précédent défini, par l'intermédiaire de la commande **pinMode**, la LED embarquée, désignée par **LED_BUILTIN**, en sortie (valeur **OUTPUT**).

L'intérêt des montages utilisant des microcontrôleurs est l'interaction avec l'environnement : les fonctions les plus utilisées sont celles qui permettent d'envoyer ou lire des informations (des tensions) sur une des broches de la carte. Ces instructions sont regroupées dans la procédure loop().

La fonction **digitalWrite** - dont la syntaxe est **digitalWrite(broche, valeur)** - permet d'écrire une donnée numérique sur la broche désignée en premier argument, celle-ci devant être au préalable réglée en sortie. Le deuxième argument indique la valeur à envoyer sur la broche. La valeur **HIGH** envoie une tension de 5 V et la valeur **LOW** envoie une tension de 0 V. Aucune autre valeur n'est possible, c'est du numérique.

Le programme précédent permet donc dans un premier temps d'allumer la LED embarquée - fonction **digitalWrite(LED_BUILTIN, HIGH)** - puis de l'éteindre - fonction **digitalWrite(LED_BUILTIN, LOW)**.

Il faut remarquer que les durées d'allumage ou d'extinction sont réglées par la fonction **delay()** : celle-ci permet d'effectuer une pause lors de l'exécution du programme dont la durée, passée en argument, est exprimée en millisecondes ; dans notre programme, la LED est allumée puis éteinte pendant une seconde.

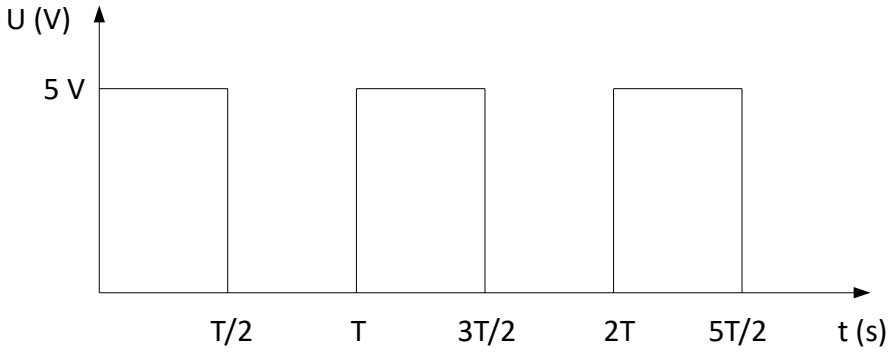
Le langage Arduino possède 3 autres fonctions d'entrée / sortie :

- ✓ **digitalRead(broche)** : mesure la valeur de la tension (5 V ou 0 V) sur la broche passée en argument. Cette broche doit être réglée en entrée. La valeur retournée est de type entier.
- ✓ **analogWrite(broche, valeur)** : écrit une donnée (valeur) sur la broche passée en argument. La broche doit être réglée en sortie et compatible avec le mode PWM. La valeur doit être de type entier et comprise entre 0 et 255.
- ✓ **analogRead(broche)** : mesure la valeur d'une tension analogique sur broche passée en argument. La broche doit être réglée en entrée. La valeur retournée est de type entier.

Il est maintenant temps de préciser (un peu ...) ce que sont les signaux PWM :

La carte Arduino ne délivre que des signaux numériques périodiques (la tension est égale à 5 V pendant une certaine fraction de la période puis elle est de 0 V pendant l'autre fraction de période).

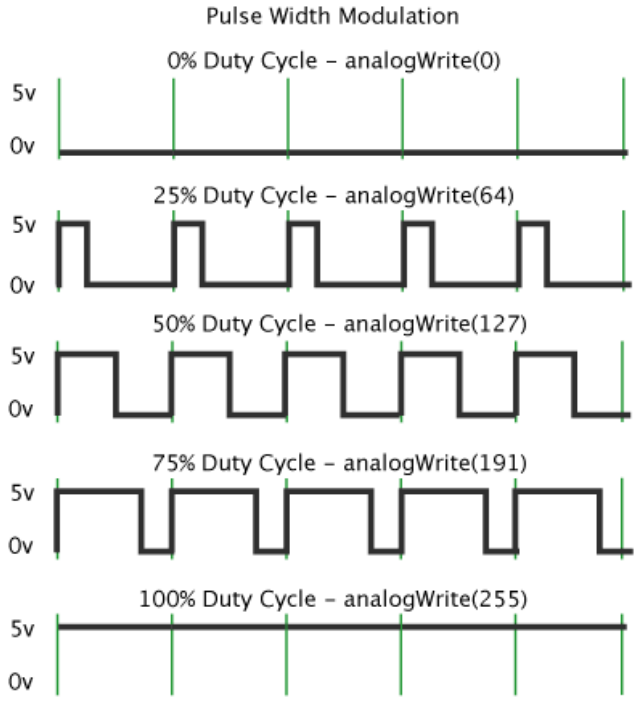
Exemple : le signal carré



Les signaux PWM sont des signaux rectangulaires de rapport cyclique variable : la tension prend la valeur 5 V pendant une durée variable en fonction de la commande envoyée à la carte par l'intermédiaire de la fonction **analogWrite()**.

Le rapport cyclique est le rapport entre la durée pendant laquelle la tension est égale à 5 V et la période ; par exemple, le signal représenté ci-dessus a un rapport cyclique de 1/2.

Exemples de signaux de différents rapports cycliques (extrait de la documentation officielle Arduino) :



Il faut remarquer que l'argument de la fonction **analogWrite()** est une valeur entière comprise entre 0 et 255, cette valeur indique le rapport cyclique en pourcentage. Un dispositif permet alors à la carte Arduino de générer une tension analogique dont la valeur est la valeur moyenne du signal numérique PWM.

Pour la lecture d'une tension en utilisant la fonction **analogRead()**, la carte Arduino utilise des CAN : la valeur de la tension est lue sur une des bornes analogique (A0 à A5) et est ensuite convertie en une valeur numérique.

Précisons un peu : le CAN d'Arduino accepte en entrée une tension comprise entre 0 V et $U_{réf}$ et fourni en sortie une valeur binaire comprise entre 0 et 1023 (le CAN utilise 10 bits en sortie, la valeur binaire est donc comprise entre 0 et 2^{10} - 1 = 1023).

Il faut remarquer que la valeur par défaut de $U_{réf}$ est 5 V mais cette valeur peut être modifiée - par l'intermédiaire de la broche AREF - afin d'améliorer la précision de la conversion. On peut par exemple relier la broche AREF à la broche 3.3 V.

Pour être utilisée dans un programme, la valeur binaire devra être convertie en une valeur décimale. Ceci peut se faire par produit en croix (règle de proportionnalité) ou en utilisant la fonction **map()**.

Exemple :

On mesure la tension sur la broche A0 ; la fonction **analogRead(0)** renvoie la valeur 458.

La valeur de la tension est donc $\frac{458 \times 5,0}{1023}$ = 2,24 V

En utilisant la fonction **map()**, il faudrait saisir :

```
valeurLue = analogRead(0) ;
tension = map(valeurLue, 0, 1023, 0, 5) ;
```

Ici, la fonction réalise la conversion d'une valeur comprise entre 0 et 1023 en une valeur comprise entre 0 et 5 V.

LE MONITEUR SÉRIE

Le port USB sert à la fois d'alimentation de la carte Arduino et à la communication avec l'ordinateur. Pour utiliser le port série afin d'afficher des informations envoyées par le programme, il est nécessaire de démarrer la liaison série en plaçant dans la partie setup du programme la commande **Serial.begin(vitesse)**. L'argument vitesse est la vitesse de communication série exprimée en bits par seconde ou encore en bauds.

La commande **Serial** crée un objet (c'est comme une variable en plus évolué ...) possédant un ensemble de propriétés, **begin()** est une de ces propriétés. Deux autres propriétés sont très utiles, il s'agit de **print()** et **println()** :

La commande **Serial.print("Hello world")** va envoyer sur le port série le texte Hello world ; la commande **Serial.println()** ferait de même mais en ajoutant un retour à la ligne.

La commande **Serial.print()** permet aussi d'envoyer des valeurs numérique formatées. Par exemple, si la variable maValeur contient la valeur 123,45, la commande **Serial.print(maValeur, 1)** affichera sur le port série 123.4.

Les informations envoyées par le programme vers le port série sont visibles en démarrant le moniteur série (bouton tout à droite de la zone 2 de la fenêtre de l'IDE). Une fenêtre indépendante s'ouvrira alors et il faudra vérifier - pour que la communication se fasse correctement - que la vitesse (en baud) indiquée en bas et à droite de cette fenêtre soit la même que celle utilisée par le programme. Au besoin, il faut régler cette vitesse dans le moniteur série en cliquant sur la flèche à droite de la vitesse.

SIMULATION DE MONTAGES : TINKERCAD

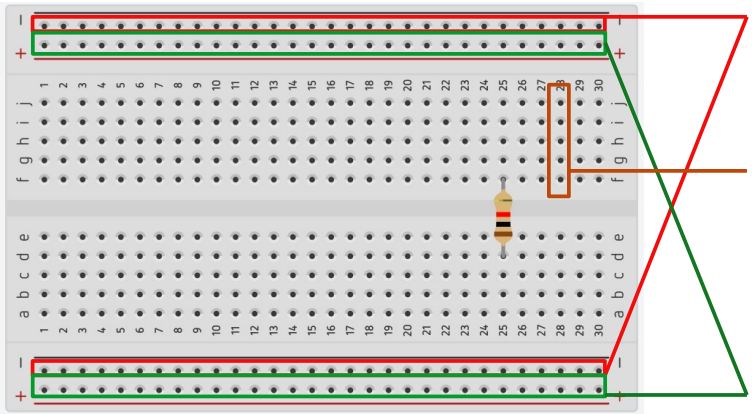
TinkerCad est une plateforme permettant la simulation de montages incluant une carte Arduino. Pour commencer à l'utiliser, il faut se rendre sur le site www.tinkercad.com et créer un compte.

Il faut ensuite cliquer (dans la partie droite de la fenêtre) sur **Circuits** puis sur **Créer un circuit**. Le circuit se réalise par de simples glisser / déposer des composants dont les symboles sont situés dans la partie droite de la fenêtre. Une connexion se réalise par un clic depuis le début de la connexion, un clic pour un "changement de direction" et clic à la fin de la connexion. Il faut remarquer que la couleur des fils peut se modifier en le sélectionnant et en choisissant la couleur ensuite. De la même façon, les caractéristiques des composants - la valeur de la résistance, par exemple - peuvent être modifiées.

Le programme pilotant la carte est saisi dans une fenêtre apparaissant après avoir cliquer sur **Code**. La programmation s'effectue par blocs comme dans Scratch, en texte ou les deux. Nous utiliserons le langage propre à l'IDE d'Arduino, la programmation en texte est donc conseillée.

Il faut remarquer que pour faire des montages réellement et virtuellement avec tinkerCad, on utilise une breadboard. C'est une plaque d'essais sans soudure sur laquelle on viendra enficher des composants et des fils afin de relier ces différents composants à la carte Arduino.

Plan de la plaque d'essai :



Ligne de connecteurs liés entre eux
(généralement reliés à la masse de la carte Arduino)

Colonne de connecteurs liés entre eux

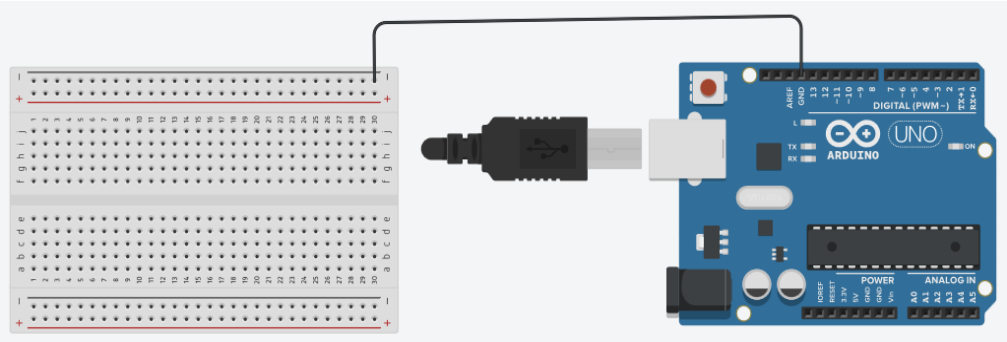
Ligne de connecteurs liés entre eux
(généralement reliés à l'alimentation +5 V de la carte Arduino)

Il faut remarquer que la breadboard est séparé par un espace central. Une colonne de connecteurs est donc séparée en deux zones de connecteurs non reliés entre eux (la résistance représentée ci-dessus n'est donc pas court-circuitée).

Exemple : Réalisation d'un circuit simple destiné à allumer et éteindre une LED Verte

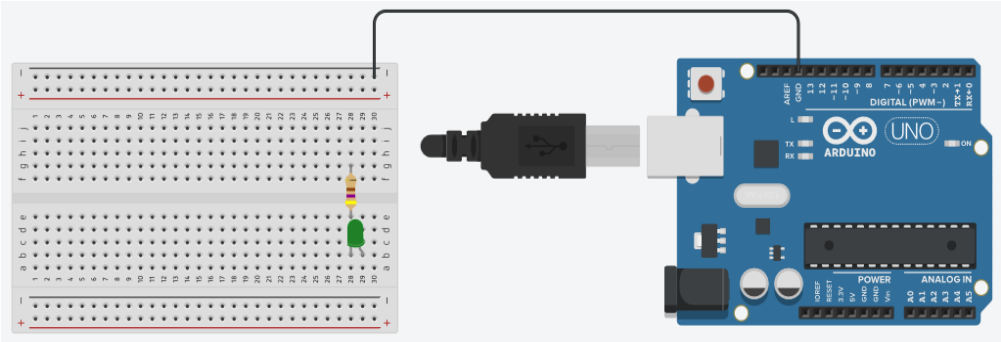
- Cliquer sur **Créer un circuit**.
- Dans la liste déroulante à droite de la fenêtre, sélectionner **Composants de base**, chercher la carte Arduino dans liste des symboles apparaissant en dessous, cliquer sur le symbole et le déplacer vers la zone de travail.
- Placer ensuite une plaquette d'essai (celle de petite taille suffira ...).
- Adapter l'affichage en cliquant sur le bouton ⛶ situé dans la partie gauche de l'affichage.
- Relier la ligne de masse de la plaque d'essai à la broche masse (GND) de la carte : cliquer sur la broche GND de la carte, cliquer pour effectuer une déviation et enfin sur un des connecteurs de la plaque d'essai.
  Il est possible de changer les propriétés de la connexion créée : choisissez la couleur noire.

À ce stade, l'affichage devrait avoir l'allure suivante :



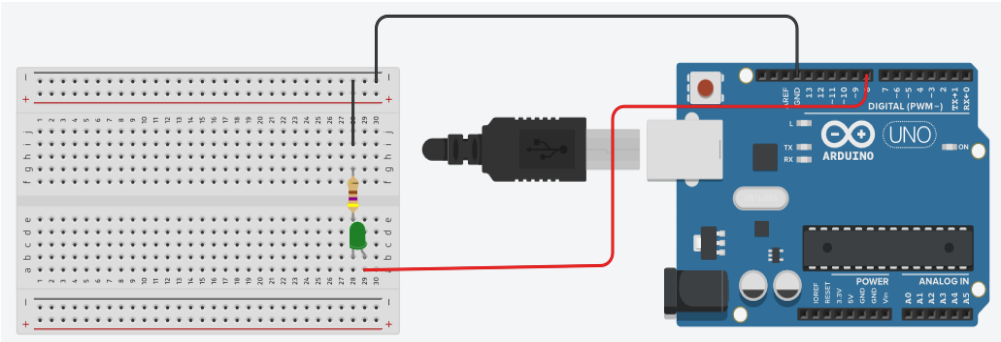
- Placer une LED verte sur la plaque d'essais ainsi qu'une résistance de 470 Ω en série.
  La LED est un composant polarisé, il faut donc respecter un sens de branchement afin qu'elle soit branchée dans sons sens passant ; connecter la résistance à la borne "droite" de la LED.

L'affichage devrait ressembler à :



- Relier maintenant, par un fil rouge, la seconde borne de la LED (celle qui est "tordue") à la sortie Digital n°8 et, par un fil noir, la seconde borne de la résistance à la ligne de masse de la plaque d'essai.

L'affichage devrait ressembler à :



Le circuit est maintenant réalisé. Passons à la programmation :

- Cliquer sur **Code** puis sélectionner l'option **Texte**.
- Effacer les commandes des blocs setup() et loop().
  Faites attention de ne pas effacer les accolades délimitant ces blocs.
- Pour celles et ceux qui sont plus à l'aise, écrire le programme qui permettra de faire clignoter la LED. Elle devra être allumée pendant une seconde et éteinte pendant la même durée.

  Pour les autres, saisir le programme donné page suivante.

```
int brocheLED = 8 ;

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

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

- Démarrer la simulation en cliquant sur **Démarrer la simulation** (en haut et à droite de la fenêtre), la diode devrait maintenant clignoter.
  Le code peut être masqué en cliquant sur **Code**.
- Quand vous serez lassé de contempler vos exploits, cliquez sur **Arrêter la simulation**.
- Vous pouvez nommer votre réalisation en cliquant sur le nom placé juste à droite du symbole TinkerCad ( ) et en saisissant le nom que vous voulez donner à votre circuit.

ÉLÉMENTS DE LANGAGE

Nous avons déjà rencontré les incontournables fonctions pinMode(), digitalWrite(), digitalRead(), analogWrite() et analogRead() ainsi que la commande delay().

Le langage Arduino possède évidemment bien plus de fonctionnalités :

✓ **Types de variables** :

int : variable de type entier (de - 32 768 à 32 767) - codée sur 2 octets

unsigned int : variable de type entier (de 0 à 65 535) - codée sur 2 octets

long : variable de type entier (de - 2 147 483 à 2 147 647) - codée sur 4 octets

unsigned long : variable de type entier (de 0 à 4 294 967 295) - codée sur 4 octets

float : variable de type réel (de - $3{,}4028235.10^{-38}$ à $3{,}4028235.10^{+38}$) avec, au total, 6 ou 7 chiffres - codée sur 4 octets

double : variable de type float avec une double précision - codée sur 4 octets

char : variable correspondant à un caractère - codée sur 1 octet

Tableau : ensemble de variables d'un type donné, chacune est repérée par un index (commençant à 0)

Exemples :

char message[7] = "bonjour" est un tableau de caractère, message[2] désigne la lettre n

int mesvaleurs[6] = {2, 4, -8, 3, 5, 6} est un tableau d'entier, mesvaleurs[0] prend la valeur 2

String : les chaînes de caractères sont des tableaux de caractères (voir ci-dessus) mais la catégorie (on dit une classe) String permet de manipuler plus facilement ces suites de caractères (pour plus de détails voir la référence en français du langage Arduino donnée en annexe).

byte : variable de type octet (entier non-signé dont la valeur est comprise entre 0 et 255)

boolean (ou bool) : variable booléenne (ne pouvant prendre que les deux valeurs true ou false)

- ✓ **Conversions de types de données** :

  Les fonction suivantes convertissent une valeur x de n'importe quel type en donnée de type indiqué.

  char(x), int(x), long(x), float(x), byte(x)

- ✓ **Fonction mathématiques** (les paramètres sont de types float) :

  abs(x) : retourne la valeur absolue du nombre x

  max(x, y) : retourne le plus grand des deux nombres x et y

  min(x, y) : retourne le plus petit des deux nombres x et y

  sq(x) : retourne le carré du nombre x

  sqrt(x) : retourne la racine carrée du nombre x

  pow(x, y) : retourne la valeur du nombre x élevé à la puissance y

  map(nombre, nombre_min, nombre_max, limite_basse, limite_haute) : retourne la valeur convertie d'un nombre (compris entre les valeurs nombre_min et nombre_max). La valeur retournée sera comprise entre limite_basse et limite_haute (voir exemple page 6)

  cos(), sin(), tan() : sans commentaire ...

  degrees(angle) : retourne la valeur en degrés d'un angle exprimés en radians

  radians(angle) : retourne la valeur en radians d'un angle exprimés en degrés

  PI : valeur du nombre pi

- ✓ **Opérateurs** :

  = : affectation d'une valeur à une variable

  +, -, *, / : réalisation des opérations mathématiques correspondantes

  % : calcule le reste d'une division d'un entier par un autre

- ✓ **Opérateurs de comparaison** :

  ==, !=, <, >, <=, >= : égalité, différence, inférieur à, supérieur à, inférieur ou égal à, supérieur ou égal à

- ✓ **Opérateurs booléens** :

  &&, ||, ! : ET booléen, OU booléen, NON booléen

- ✓ **Opérateurs composés** :

  ++ : incrémentation de un

  Exemples :

  si x vaut 2, y = x++ équivaut à y = 3 et x garde la valeur 2

  si x vaut 2, y = ++x équivaut à y = 3 et x prend aussi la valeur 3

  -- : décrémentation de un

  +=, -=, *=, /= : les opérateurs composés réalisent l'opération et l'affectation

  Exemples :

  x += y équivaut à x = x + y

  si x vaut 2, x *= 10 vaut 20 (équivalent à x = x * 10)

✓ **Commandes de contrôle** :

if (...) : teste si une condition est vraie et exécute le bloc d'instructions suivant

if (...) else ... : teste si une condition est vraie et exécute le bloc d'instructions suivant ; si la condition est fausse, le bloc d'instruction suivant **else** est exécuté.

for (initialisation ; condition ; incrementation) : exécute le bloc d'instruction suivant si la condition est réalisée, réalise aussi l'incrémentation

Exemple : for (int i = 0 ; i < 20 ; i++) exécute 20 fois un bloc d'instructions en incrémentant i

switch (valeur) case ... : un exemple sera meilleur qu'un long discours ...

```
switch (valeur) {
  case 1 :
  bloc d'instructions à effectuer si valeur = 1
  break ;
  case 2 :
  bloc d'instructions à effectuer si valeur = 2
  break ;
  default :
  si aucune des conditions précédentes n'est réalisées
}
```

while (expression) : exécute le bloc d'instructions suivant tant que l'expression est vérifiée

Si l'expression est initialement fausse, le bloc d'instruction n'est pas être exécuté (le test est effectué en premier)

do {...} while (expression) : exécute le bloc d'instructions suivant **do** tant que l'expression est vérifiée. Le bloc d'instruction est toujours exécuté même si l'expression est fausse (le test est effectué en dernier)

break : instruction permettant de sortir d'une boucle for, while, do ... while

return : termine une fonction en renvoyant une valeur calculée par cette fonction

Exemple de fonction :

```
int multiplication(int x, int y) {
  int resultat ;
  resultat = x * y ;
  return resultat ;
}
```

Cette fonction réalise la multiplication de deux entiers passés en arguments de la fonction et renvoie leur produit.

void : mot-clé utilisé pour déclarer une fonction qui exécutera des instructions mais qui ne retournera pas de résultat. Ce type de fonction est appelée procédure.

Exemple de fonction :

```
void allumeLed (int valeur) {
  digitalWrite{led_0, LOW) ;
  if (valeur == 1) {
   digitalWrite(led_0, HIGH) ;
}
```

Cette fonction éteint une LED puis l'allume si la valeur passée en argument est égale à 1.

- ✓ **Quelques fonctions utiles** :

  tone(broche, fréquence, durée) : génère un son dont la fréquence et la durée sont passées en paramètres sur la broche indiquée (la durée est optionnelle)

  noTone(broche) : stoppe la génération d'un son créé par la fonction **tone**

  millis() : retourne la durée écoulée en millisecondes depuis le début du programme

- ✓ **Commenter le programme** :

  // : permet d'insérer du texte en commentaire sur une ligne

  /* et */ : permet d'insérer des commentaires sur plusieurs lignes (la marque de début est /* et celle de fin est */)

LES LIBRAIRIES

Les capacités matérielles d'une carte Arduino peuvent être étendues par l'ajout de cartes dédiées (des shields) : on trouve des cartes permettant d'ajouter un GPS au montage, d'écrire des données sur une carte SD, etc ...

Le langage Arduino est aussi extensible par l'ajout de commandes. Ces nouvelles commandes sont regroupées dans des librairies permettant souvent de piloter facilement les extensions matérielles.

Une de ces librairies, déjà rencontrée, est la librairie Serial permettant de piloter la communication entre la carte Arduino et un ordinateur via le port série.

Souvent les librairies "classiques" sont incorporées au langage Arduino, il suffit d'utiliser les commandes disponibles (voir par exemple l'utilisation de la bibliothèque Serial décrite page 6). D'autres ne le sont pas et il est alors nécessaire de les insérer dans un programme, après les avoir éventuellement installées dans l'environnement, par l'intermédiaire de la commande **#include**. Ceci peut se faire dans le logiciel en utilisant la commande **Inclure une bibliothèque** du menu **Croquis**.

Exemple :

La commande **#include <LiquidCrystal.h>** permet d'inclure dans un programme Arduino et d'utiliser, des commandes permettant l'affichage de caractères sur un afficheur LCD alphanumérique.

Pour avoir plus de renseignements à propos des librairies disponibles et leur éventuelle installation, consultez le lien donné en annexe à propos des librairies.

ANNEXES

QUELQUES PRÉCAUTIONS ...

Afin de ne pas détériorer la carte Arduino, il ne faut pas que la carte débite un courant d'intensité supérieure à 40 mA par sortie et pas plus de 200 mA au total : 6 sorties qui débiteraient 35 mA chacune endommageraient la carte.

Il ne faut jamais relier une sortie à la masse, relier deux sorties entre elles ou encore mettre une tension supérieure à 5 V sur une entrée de la carte.

RÉFÉRENCES

Arduino : premier pas en informatique embarquée (très bon pdf de 631 pages)

https://eskimon.fr/extra/ebooks/arduino-premiers-pas-en-informatique-embarquee.pdf

Références du langage Arduino (en français) :

http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Reference

Les librairies de référence (en français) :

http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Librairies

Site de référence d'Arduino (en anglais) :

https://www.arduino.cc/reference/en/