Présentation du fonctionnement du codage Arduino

V. Lacharnay
Lycée Gustave Eiffel, Dijon

2018–2019

Plan du cours

1 2 3 4 5 6 7

Syntaxe du langage

Syntaxe du langage
Code initial

La syntaxe d'un langage de programmation est l'ensemble des règles d'écriture liées à ce langage.
Avec Arduino, nous devons utiliser un code minimal lorsque l'on crée un programme. Ce code permet de diviser le programme que nous allons créer en deux grosses parties.

```
// fonction d'initialisation de la carte
void setup()
{
// contenu de l'initialisation
}

// fonction principale, elle se répète (s'éxecute) àl'infini
void loop()
{
// contenu de votre programme
}
```

Syntaxe du langage
Code initial

```
// fonction d'initialisation de la carte
void setup()
{
// contenu de l'initialisation
}
```

Cette fonction `setup()` est appelée une seule fois lorsque le programme commence. C'est pourquoi c'est dans cette fonction que l'on va écrire le code qui n'a besoin d'être exécuté une seule fois. On appelle cette fonction : fonction d'initialisation. On y retrouvera :

- la mise en place des différentes sorties ;
- l'initialisation de votre système ;
- toutes les programmations que vous ne voulez voir exécutées qu'une fois.

Syntaxe du langage
Code initial

```
// fonction principale, elle se répète (s'éxecute) àl'infini
void loop()
{
// contenu de votre programme
```

C'est donc dans cette fonction `loop()` où l'on va écrire le contenu du programme. Il faut savoir que cette fonction est appelée en permanence, c'est-à-dire qu'elle est exécutée une fois, puis lorsque son exécution est terminée, on la ré exécute et encore et encore. On parle de boucle infinie.

Syntaxe du langage
Syntaxe du codage Arduino

Il est indispensable pour faire fonctionner le codage d'implémenter des instructions. Ces dernières sont des lignes de codes qui envoient la demande de l'utilisateur au système.
Règles d'écriture en Arduino :

- les points virgules doivent terminer les instructions

```
digitalWrite(led_rouge_feux_1, HIGH);
```

- Les accolades sont les « conteneurs » du code du programme. Elles sont propres aux fonctions, aux conditions et aux boucles. Les instructions du programme sont écrites à l'intérieur de ces accolades ;
- les commentaires sont des lignes de codes non utilisées lors de la compilation, mais indispensables à la compréhension de la rédaction du code

```
// voilà un commentaire
```

Les accents sont non utilisables sauf dans les commentaires.

1 2 3 4 5 6 7

Variables

Variables
Définition des variables

Une **variable est un nombre** stocké dans un espace de la mémoire vive du microcontrôleur.

```
x=2;
```

Il existe des contraintes sur l'appellation de la variable :

- les caractères non acceptés dans le nom de la variable sont :
  - le point ;
  - la virgule ;
  - les accents.
- son nom ne doit pas commencer par un nombre.

L'écriture fournie précédemment n'est pas correcte, car il manque le type de variable souhaitée.

```
char x=2;
```

Cela permet de ne pas trop encombrer la mémoire vive de votre ordinateur.

Variables
Définition des variables

Le tableau ci-dessous indique les types de variables que vous serez amené à utiliser.

Type	Domaine	Étendue décimal	Étendue en bits
char	$\in \mathbb{N}$	-128 à +127	8 bits
int	$\in \mathbb{N}$	-32 768 à +32 767	16 bits
long	$\in \mathbb{N}$	-2 147 483 648 à +2 147 483 647	32 bits
float	$\in \mathbb{R}$	$-3.4 \cdot 10^{38}$ à $+3.4 \cdot 10^{38}$	32 bits
boolean	$\in \mathbb{R}^+$	0 à 1	1 bit

TABLE 1 – Différents types de variables utiles

Cela est utile afin de ne pas utiliser un surplus de mémoire interne.

Variables
Variable booléennes

Dans le langage Arduino, il est posible d'écrire des éléments qui pourront être utilisés comme booléen de trois manière différentes :

- Codage classique :

```
// variable est fausse, car elle vaut FALSE, du terme anglais "faux"
boolean variable = FALSE;
// variable est vraie, car elle vaut TRUE, du terme anglais "vrai"
boolean variable = TRUE;
```

- Codage avec entier :

```
// variable est fausse, car elle vaut 0
int variable = 0;
// variable est vraie, car elle vaut 1
int variable = 1;
// variable est vraie, car sa valeur est différente de 0
int variable = 42;
```

- Codage de type Arduino :

```
// variable est àl'état logique bas (= traduction de "low"), donc 0
int variable = LOW;
// variable est àl'état logique haut (= traduction de "high"), donc 1
int variable = HIGH;
```

Variables
Liste des opérations (1/2)

Il existe des applications utilisables (comme sous Python) entre les variables définies ci-dessous :

```
// définition de la variable a et assignation a la valeur 15
int a = 15;
int y = 10;
float z = 0;
```

- les symboles associés aux opérations classiques sont semblables à celles utilisées en Python : `+,-,*,/,\%` ;

```
// définition de la variable d et assignation a la valeur souhaitée
d=a+43;
// d vaut donc 58
```

- attention au type utilisé pour vos paramètres, le résultat peut être faussé :

```
y=a/2;
// y vaut alors 7, car il est défini comme un entier !
```

Variables
Liste des opérations (2/2)

Il existe des applications utilisables (comme sous Python) entre les variables définies ci-dessous :

- incrémentation :

```
y++;
// y vaut alors 8, car équivalent a y=y+1
```

- opérations similaires : `- -`, `+=`, `-=`, `*=`, `/=`
- opération de bascule

```
boolean x = 0;
x = 1 - x;
```

Sur une simulation en loop on obtient :

- Premier temps : $x = 1 - x$ soit $x = 1 - 0$ donc $x = 1$;
- Second temps : $x = 1 - x$ or x vaut maintenant 1 donc $x = 1 - 1$ soit $x = 0$;
- Troisième temps : x vaut 0 donc $x = 1 - 0$ soit $x = 1$.

La fonction not ($=!$) donne le même résultat :

```
x=!x;
```

1 2 3 4 5 6 7

Conditions

Les conditions
Types de conditions

Comme avec le langage Python, il est possible d'imposer des conditions afin de tester les variables. Les symboles utilisés pour ces conditions sont les suivantes :

- `==` : est égale à
- `<` : est inférieur à
- `>` : est supérieur à
- `<=` : est inférieur ou égale à
- `>=` : est supérieur ou égale à
- `=!` : est différent de

Pour utiliser ces conditions, il existe trois possibilités :

- `if` : représente la condition « si » ;
- `elseif` : représente la condition « si pas la première, mais la deuxième » ;
- `else` : représente la condition « sinon ».

Les conditions

Exemple

```
int noteSur20 = 15;
int tempsDeTravail = 3;

if(noteSur20 < 10)
{
qualiteDuTravail = FALSE;
}
else if(noteSur20 > 10 && noteSur20 < 15 && tempsDeTravail > 2)
{
qualiteDuTravail = TRUE;
rapiditeDuTravail = FALSE;
}
else if(noteSur20 > 10 && noteSur20 < 15 && tempsDeTravail < 2)
{
qualiteDuTravail = TRUE;
rapiditeDuTravail = TRUE;
}
else
{
// le travail est excellent, rien a dire !
}
```

1 2 3 4 5 6 7

Les boucles

Les boucles

Il existe deux types principaux de boucles :

- la boucle conditionnelle , qui teste une condition et qui exécute les instructions qu'elle contient tant que la condition testée est vraie ;
- la boucle de répétition , qui exécute les instructions qu'elle contient, un nombre de fois prédéterminé.

Les boucles
La boucle `while`

Le boucle `while`, signifie `tant que`, d'un point de pue générale elle est appelée comme ça :

```
while(/* condition àtester */)
{
    // les instructions entre ces accolades sont répétées
    // tant que la condition est vraie
}
```

Exemple (Compteur)

```
// variable compteur qui va stocker le nombre de fois que la boucle
int compteur = 0;
// aura été exécutée

// tant que compteur est différent de 5, on boucle
while(compteur != 5)
{
    compteur++; // on incrémente la variable compteur a chaque boucle
}
```

Les boucles
La boucle `for`

Le boucle `for` impose une connaissance de trois paramètres :

- la création et l'assignation de la variable à une valeur de départ ;
- suivit de la définition de la condition à tester ;
- suivit de l'instruction à exécuter.

```
for(int compteur = 0; compteur < 5; compteur++)
{
    // code a executer
}
```

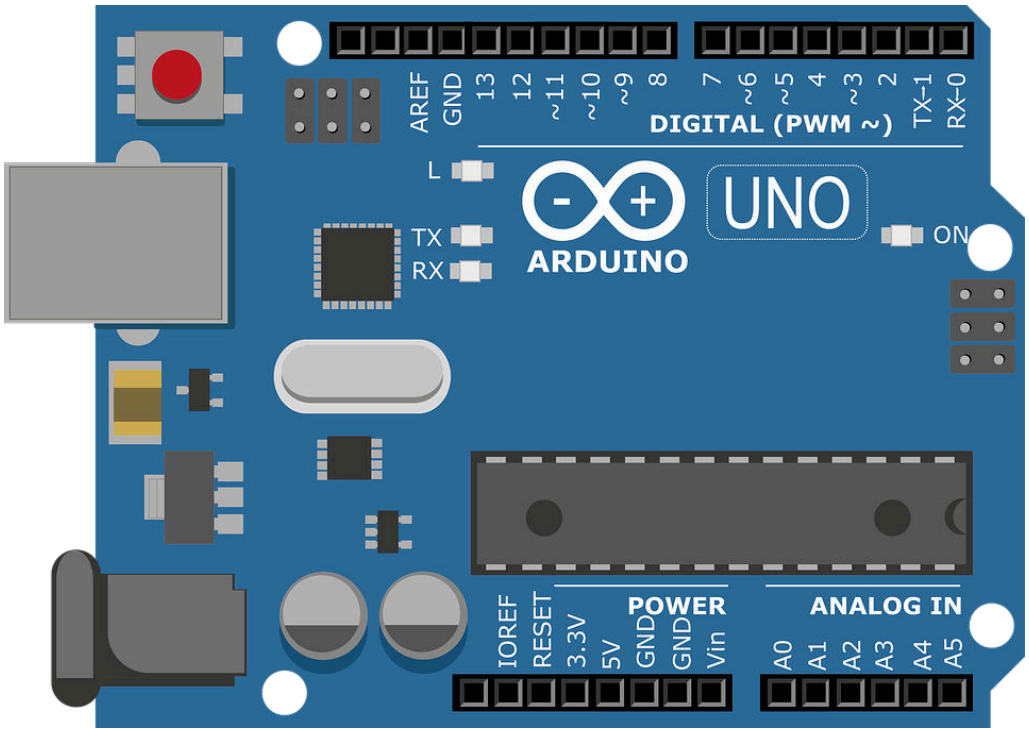
Il est possible de voir une documentation plus fournie sur le site d'origine de ces données : `https://eskimon.fr/tuto-arduino-105-le-langage-arduino-12`

1 2 3 4 5 6 7

Utilisation de la carte Arduino UNO

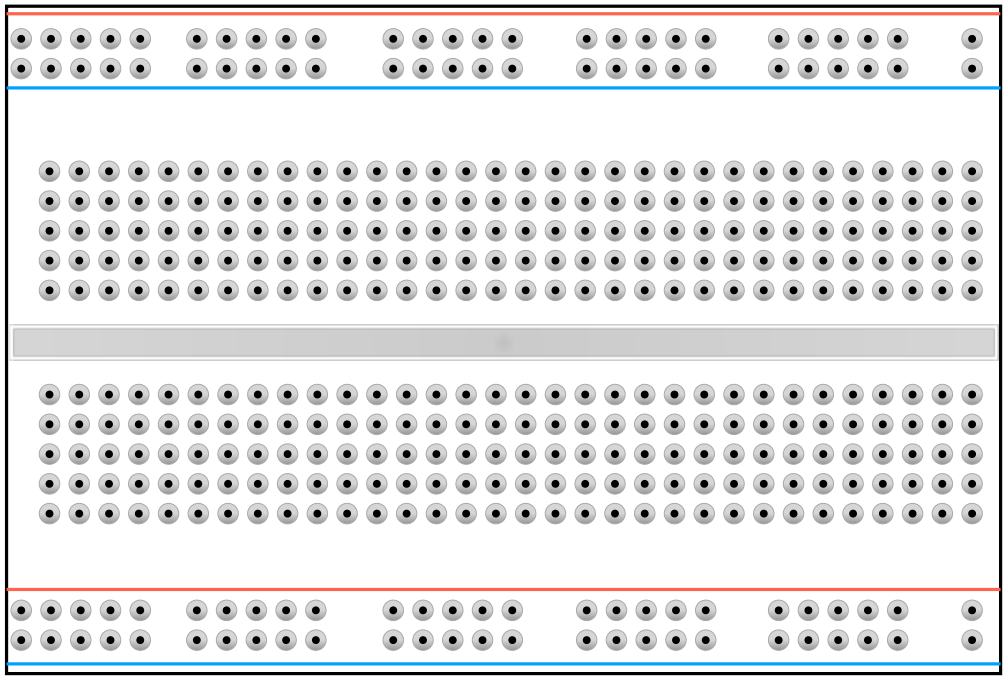
La carte Arduino UNO

La carte de prototypage est un « ordinateur » simplifié. Seul, il n'est pas possible d'interagir avec elle. La construction des circuits adaptés permettra d'envoyer et de recevoir des informations.



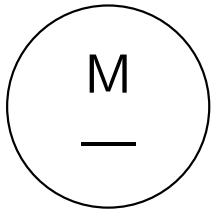
Breadboard

Il s'agit d'une platine sur laquelle on construit les circuits électroniques. Cette plaque à trous possède des lignes de connexion permettant de relier les différentes voies utilisées.



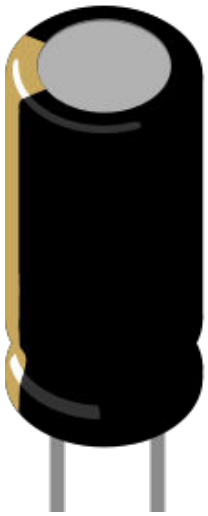
Moteur à courant continu

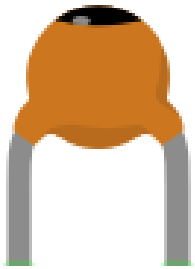
Il permet de convertir une énergie électrique en énergie mécanique de rotation. Si le sens du courant est inversé, le moteur tournera dans l'autre sens.

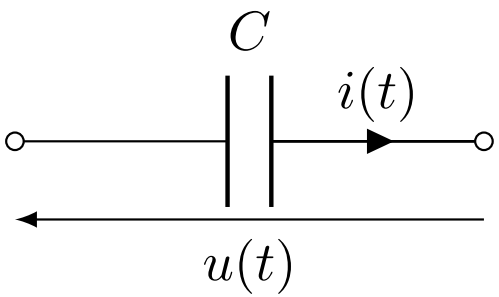


Condensateur

Ils permettent de stocker de l'énergie en se chargeant lorsque la tension du circuit est plus haute que la sienne et en se relâchant dans le cas contraire. Cela permet d'éviter les fluctuations discontinues de tensions (pour ce qui est de la continuité en courant, l'élément utilisé est une bobine).

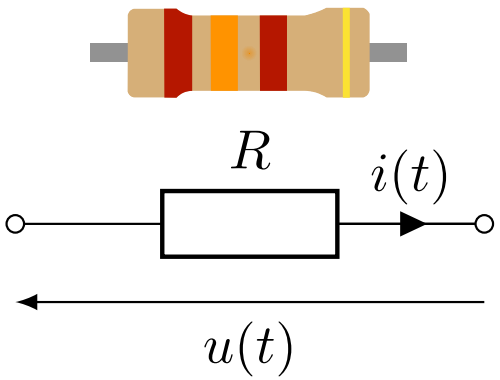


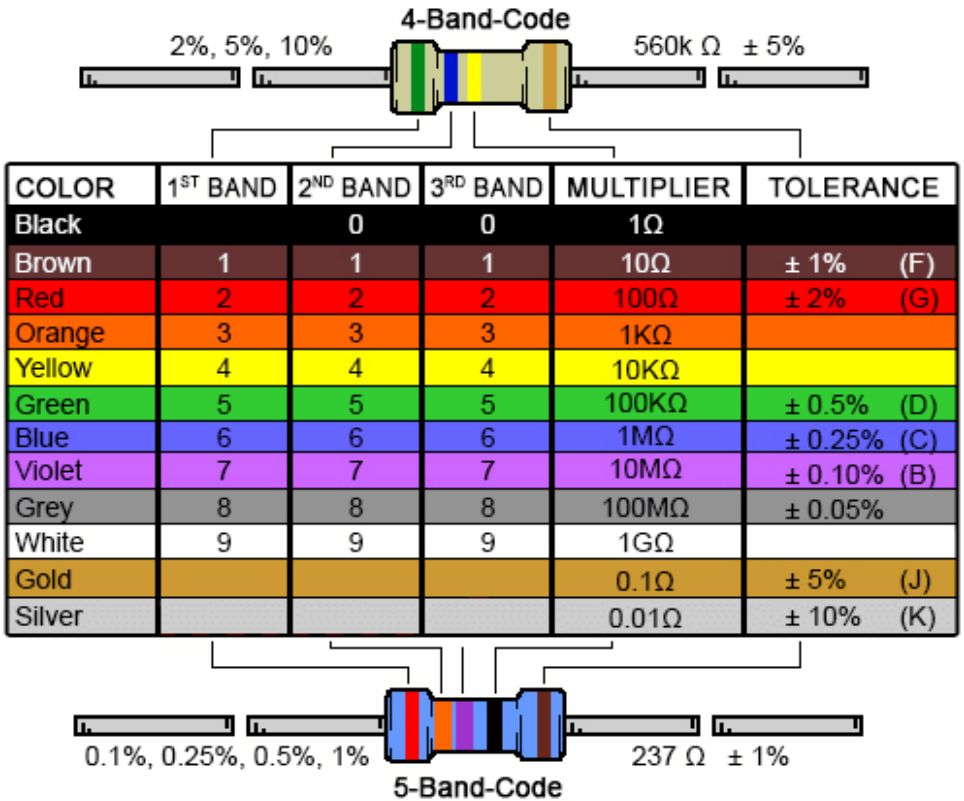




Résistance

Elle résiste au passage du courant électrique dans un circuit, en affectant la tension et le courant. Les valeurs des résistances sont mesurées en Ohms (Ω). Les bandes de couleurs sur les côtés des résistances indiquent leur valeur.

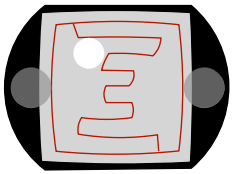


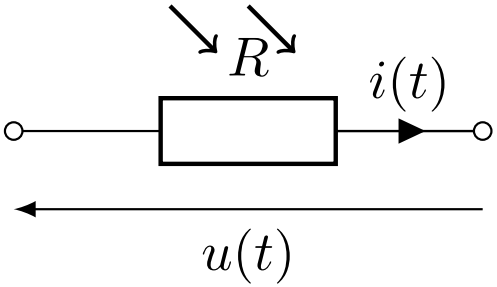


COLOR	1ST BAND	2ND BAND	3RD BAND	MULTIPLIER	TOLERANCE
Black		0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8	100MΩ	± 0.05%
White	9	9	9	1GΩ	
Gold				0.1Ω	± 5% (J)
Silver				0.01Ω	± 10% (K)

Photo-résistance

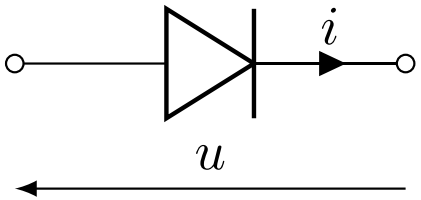
Résistance variable dont la résistance varie en fonction de la quantité de lumière qui l'éclaire.





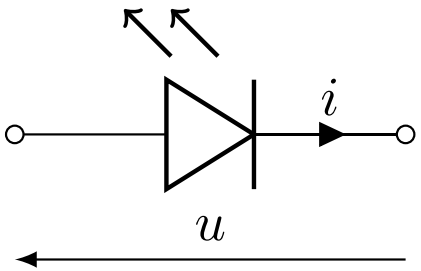
Diode

Elle permet de s'assurer que le courant ne circule que dans un sens. Ces dernières sont polarisées ce qui impose de d'être vigilant sur le sens du montage.



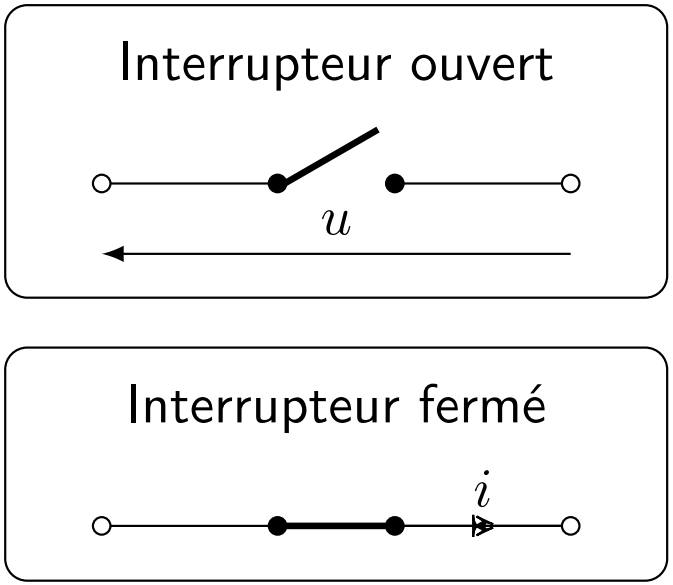
Diode électroluminescente (LED)

Type de diode qui s'illumine lors du passage du courant. Comme pour les diodes « générales » présentées précédemment, l'électricité ne peut circuler que dans un sens. L'anode, qui est classiquement reliée à l'alimentation, est la patte la plus longue et la cathode et la patte la plus courte.



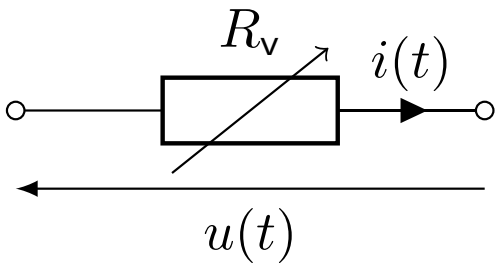
Bouton poussoir

Interrupteur momentané qui ferme un circuit lorsqu'on le presse. Il s'enfiche dans une breadboard facilement.



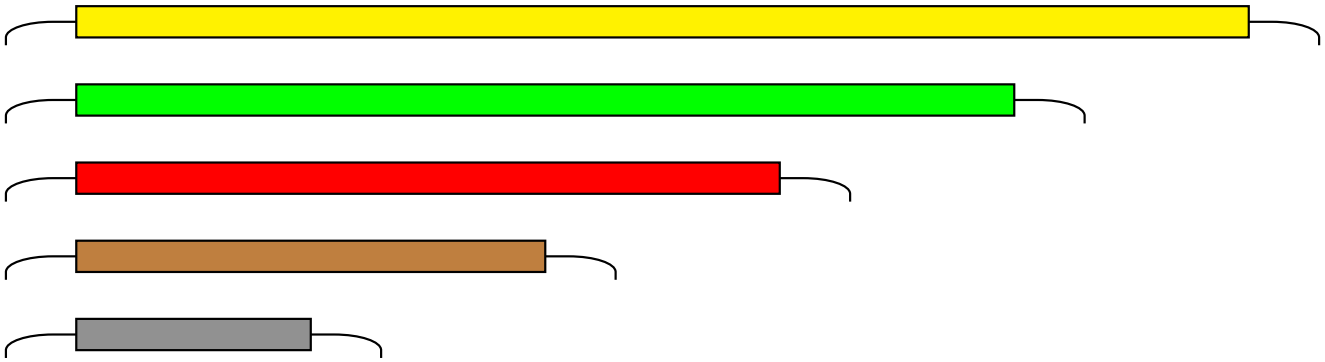
Potentiomètre

Il est équivalent à une résistance variable avec trois pattes dont deux sont reliées aux extrémités d'une résistance fixe. La patte du milieu (aussi appelé curseur) se déplace le long de la résistance, en la divisant en deux parties. Le potentiomètre permet donc de modifier la résistance utile.



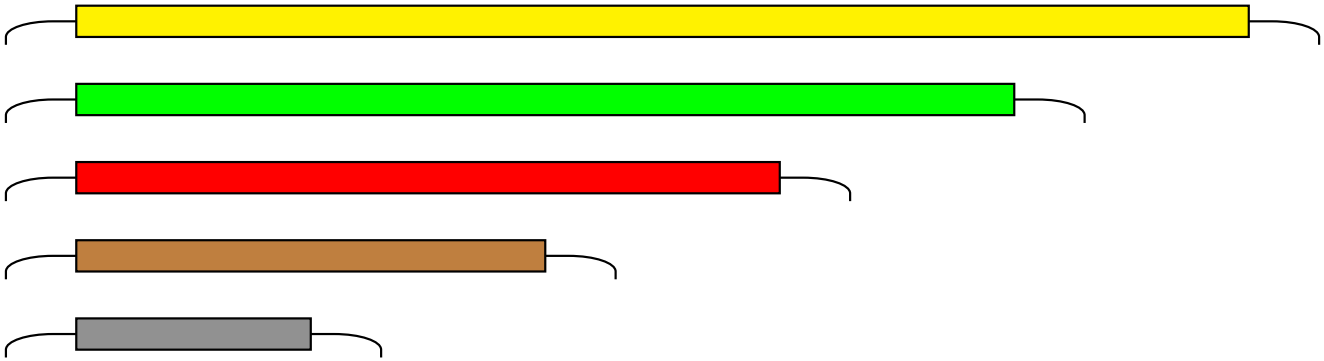
Câbles de prototypage

Ils permettent de connecter les composants, soit ensemble, soit en passant par la breadboard.



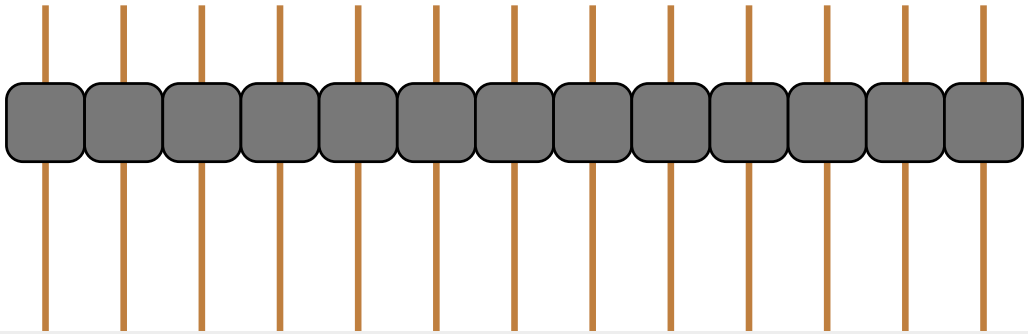
Câbles de prototypage

Ils permettent de connecter les composants, soit ensemble, soit en passant par la breadboard.



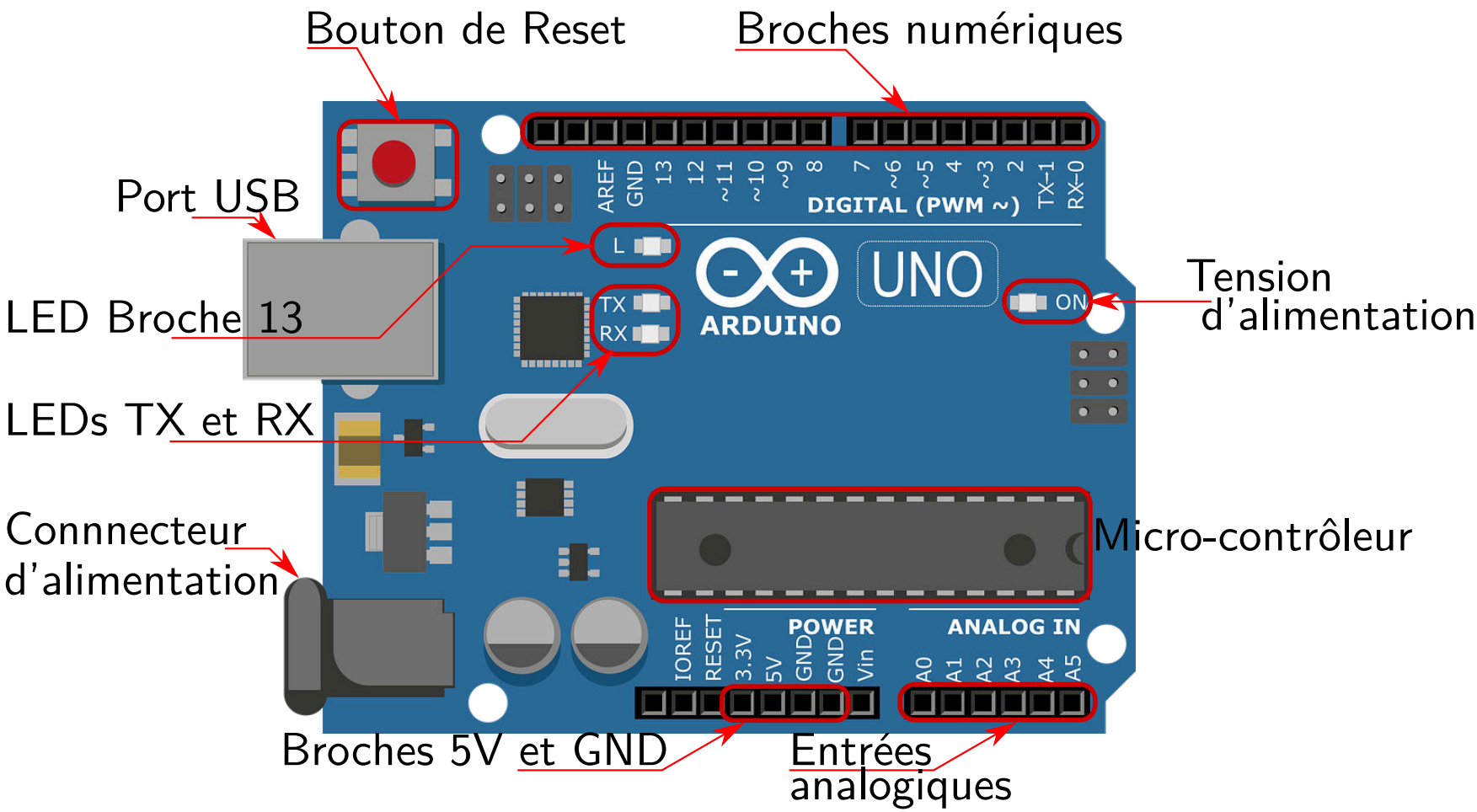
Connecteurs mâles

Ces connecteurs rentrent dans les connecteurs femelles, comme ceux d'une breadboard. Ils permettent de brancher des objets plus facilement.



Utilisation de la carte Arduino UNO

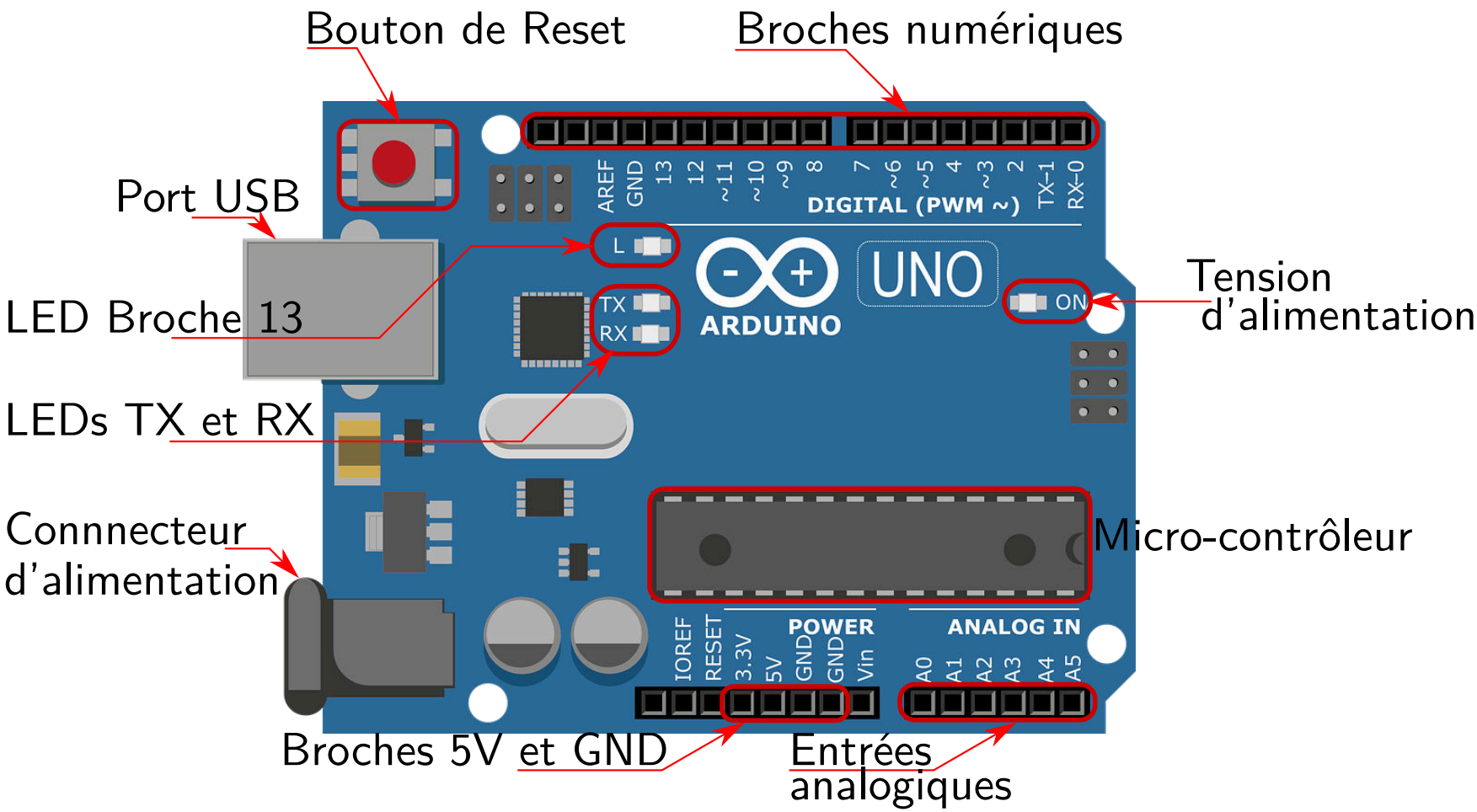
Analyse détaillée de la carte Arduino Uno



Connecteur d'alimentation : si votre carte Arduino n'est pas reliée par le port USB, il est possible de l'alimenter entre 7 et 12 Volts.

Utilisation de la carte Arduino UNO

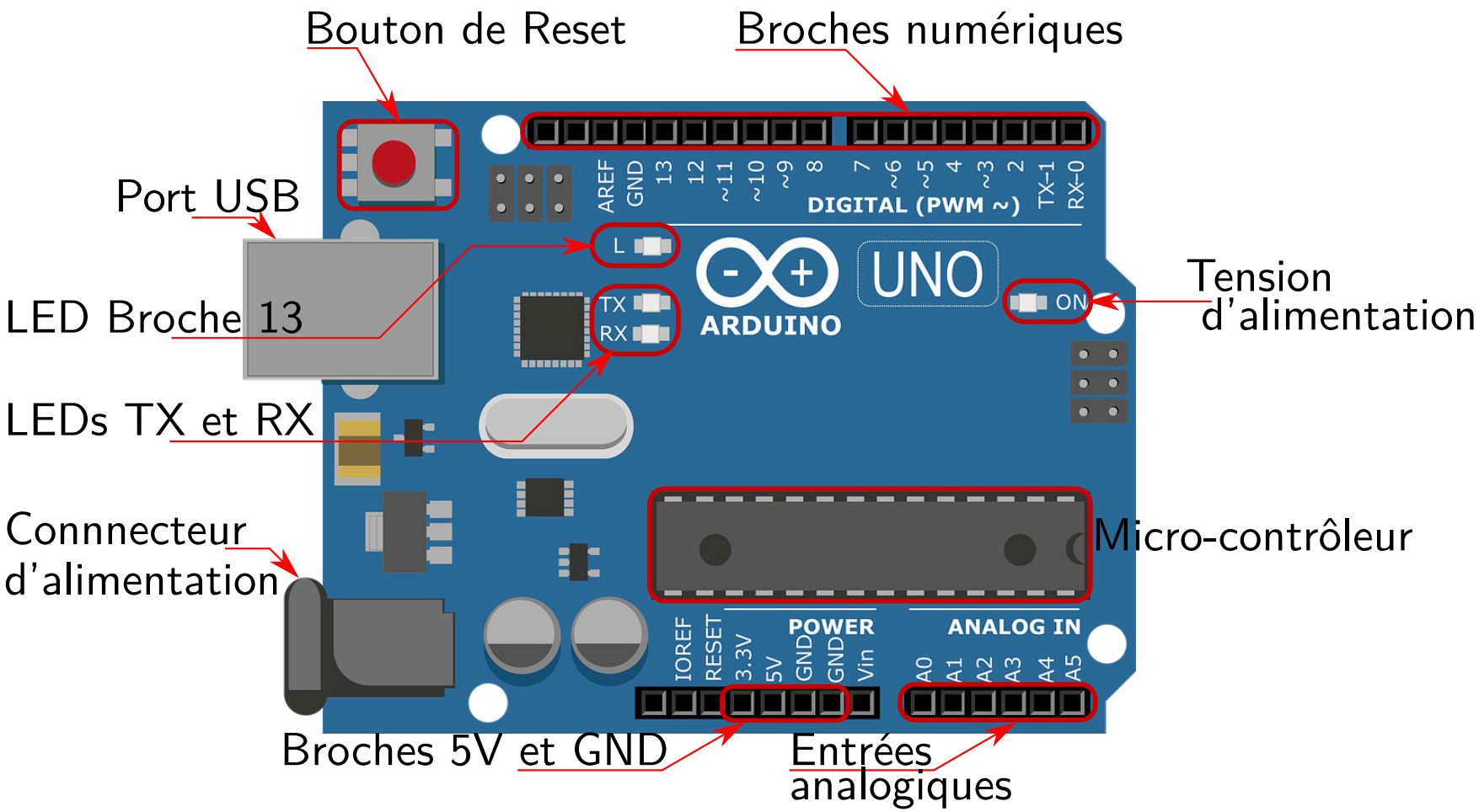
Analyse détaillée de la carte Arduino Uno



Port USB : alimentation et transfert de programmes avec code du type : `Serial.println()`

Utilisation de la carte Arduino UNO

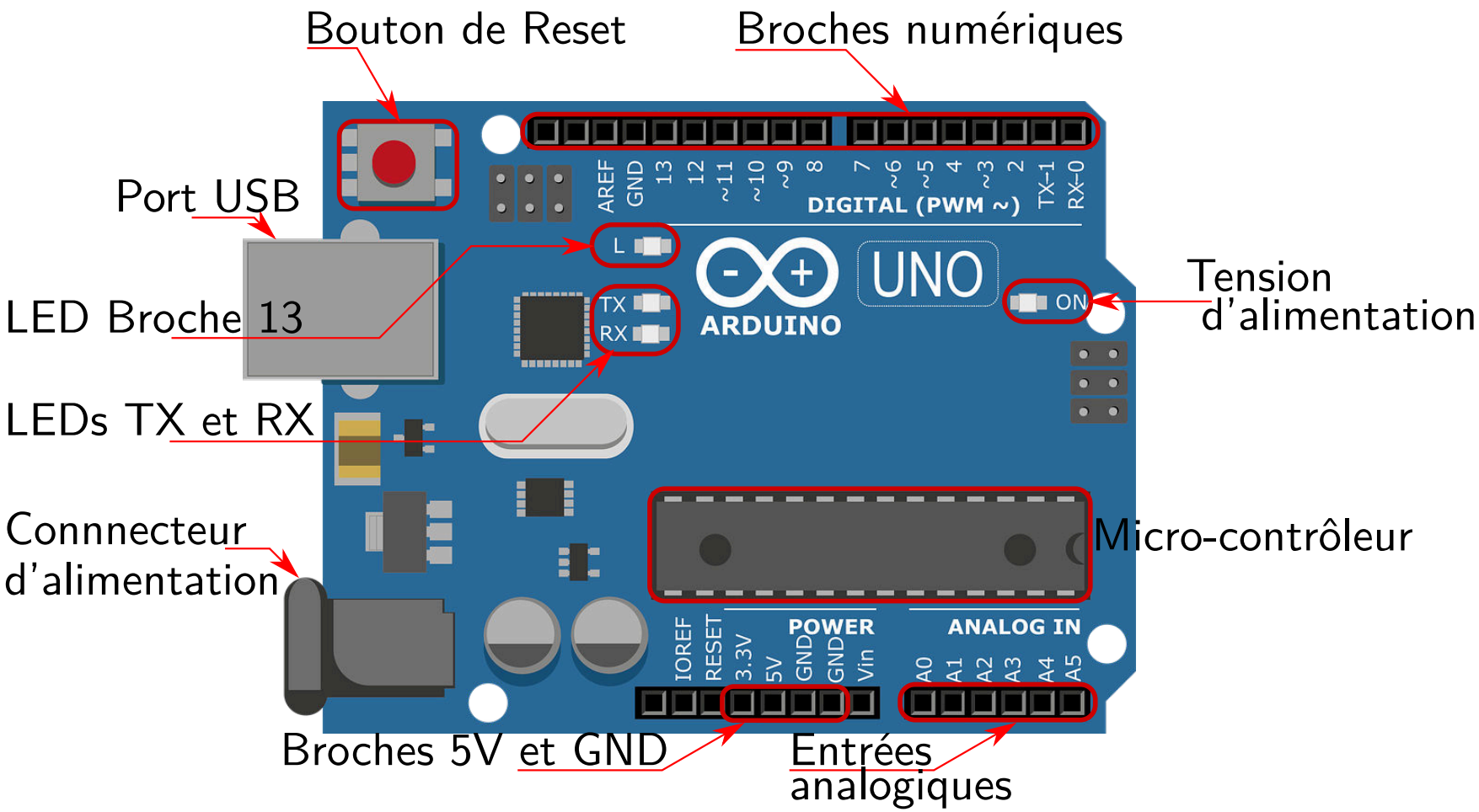
Analyse détaillée de la carte Arduino Uno



Bouton de reset : redémarre le microcontrôleur.

Utilisation de la carte Arduino UNO

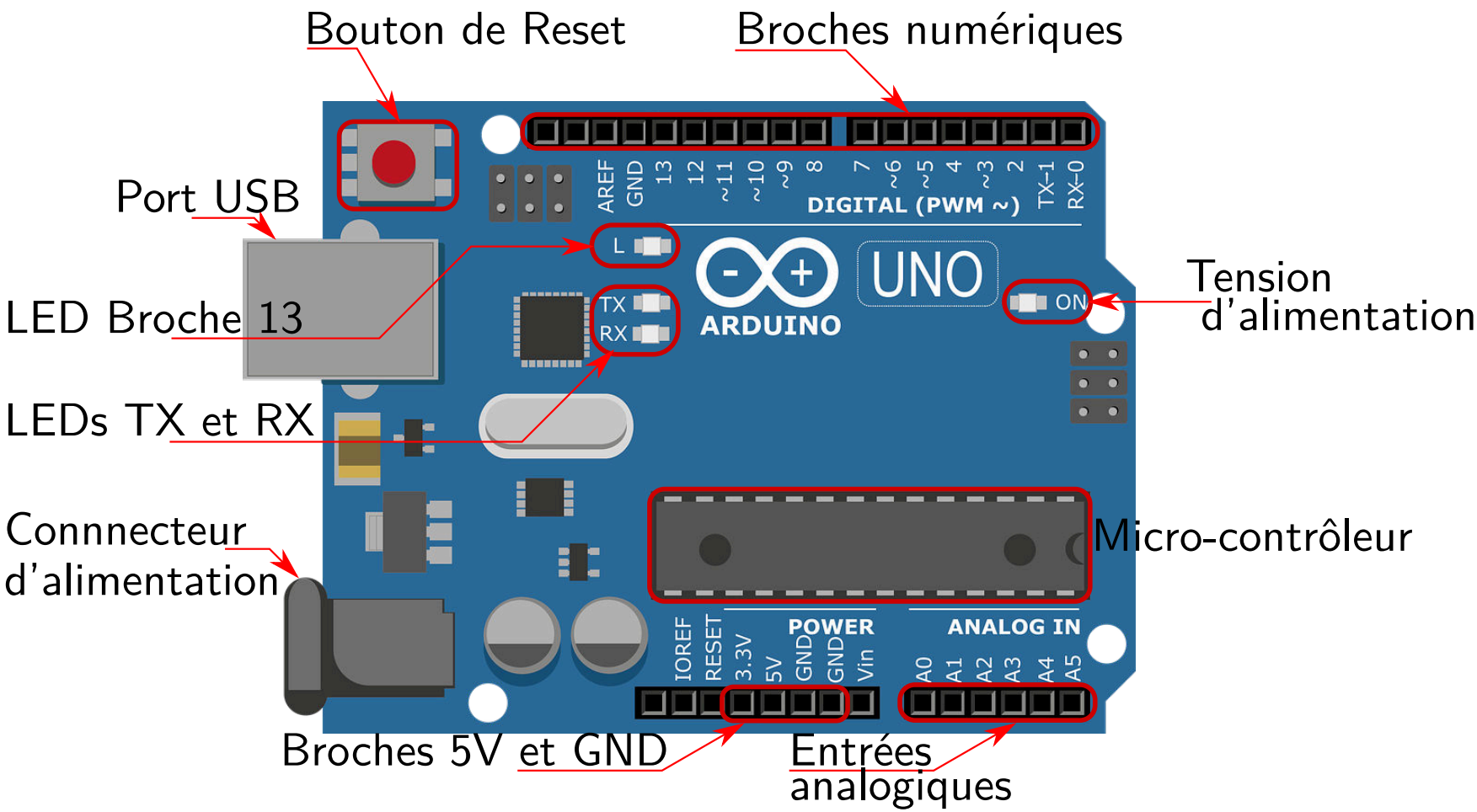
Analyse détaillée de la carte Arduino Uno



Leds TX et RX : indique la communication entre Arduino et l'ordinateur. Utile pour le débug.

Utilisation de la carte Arduino UNO

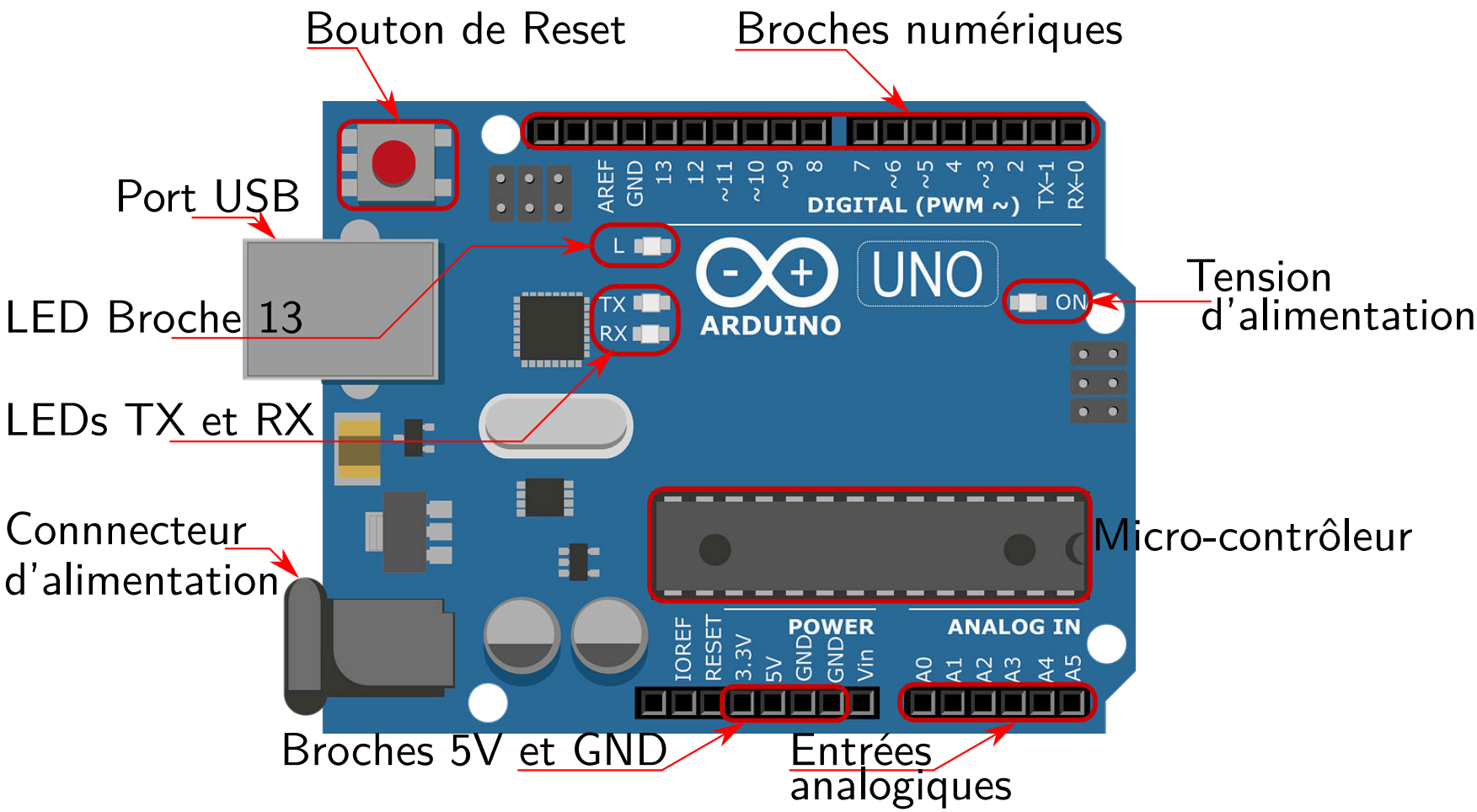
Analyse détaillée de la carte Arduino Uno



Broches numériques : entrées/sorties numériques, dont 6 sorties PWM (avec ~). Les codes utilisables sont : `digital.Read()`, `digital.Write()` (pour tous) et `analogue.Write()` (pour les broches PWM).

Utilisation de la carte Arduino UNO

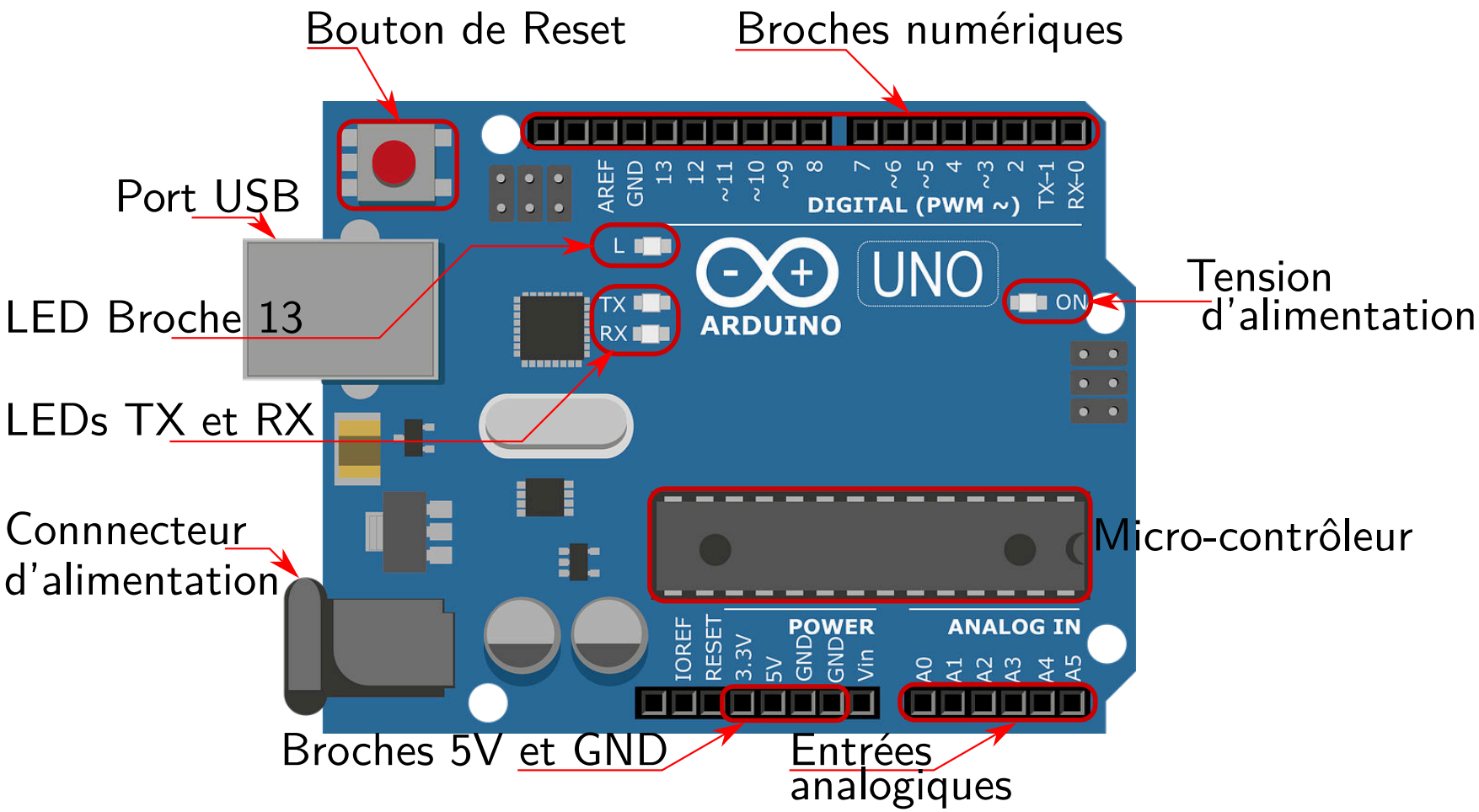
Analyse détaillée de la carte Arduino Uno



LED Broche 13 : utile pour le débug.

Utilisation de la carte Arduino UNO

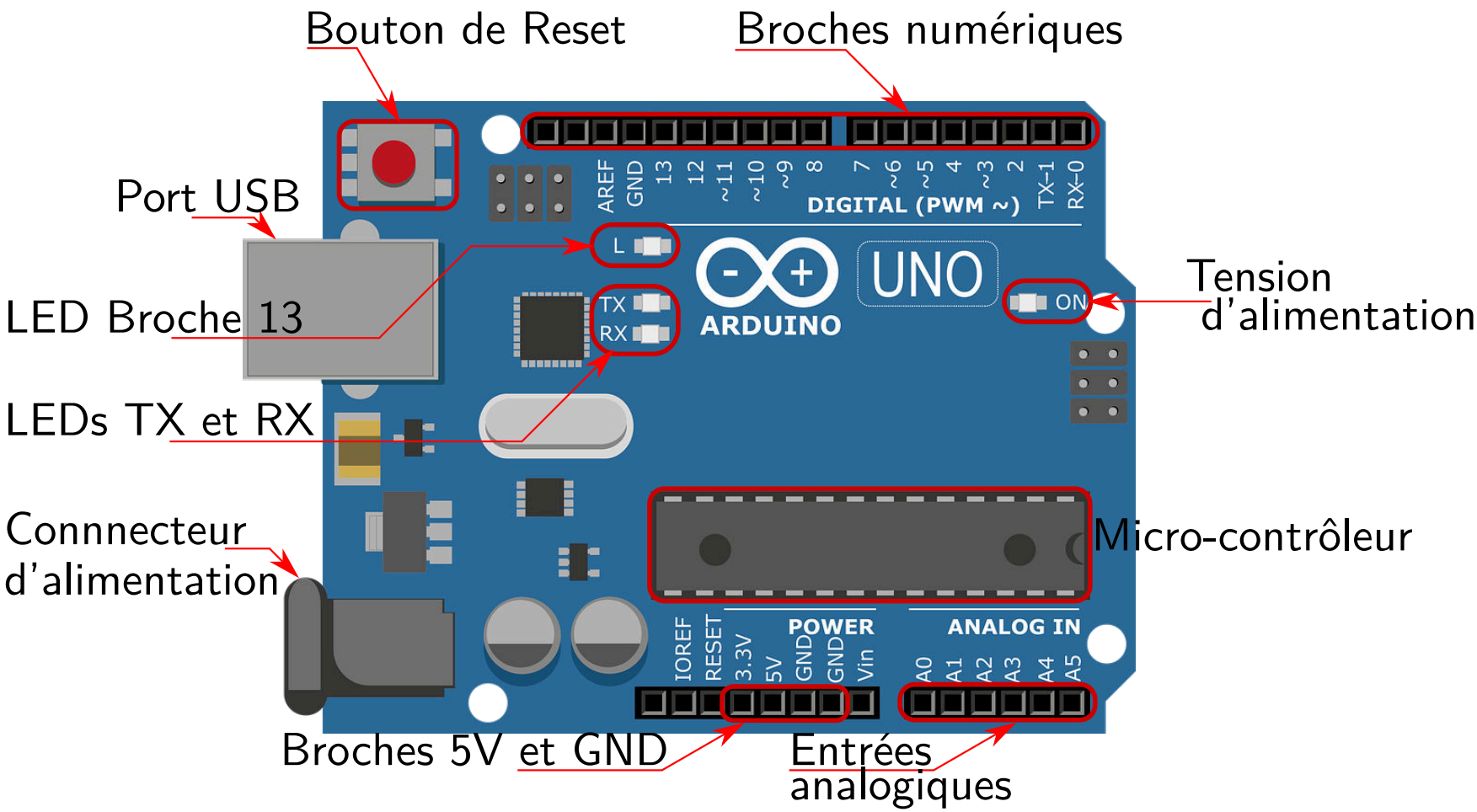
Analyse détaillée de la carte Arduino Uno



Tension d'alimentation : indique que votre carte Arduino est bien alimentée.

Utilisation de la carte Arduino UNO

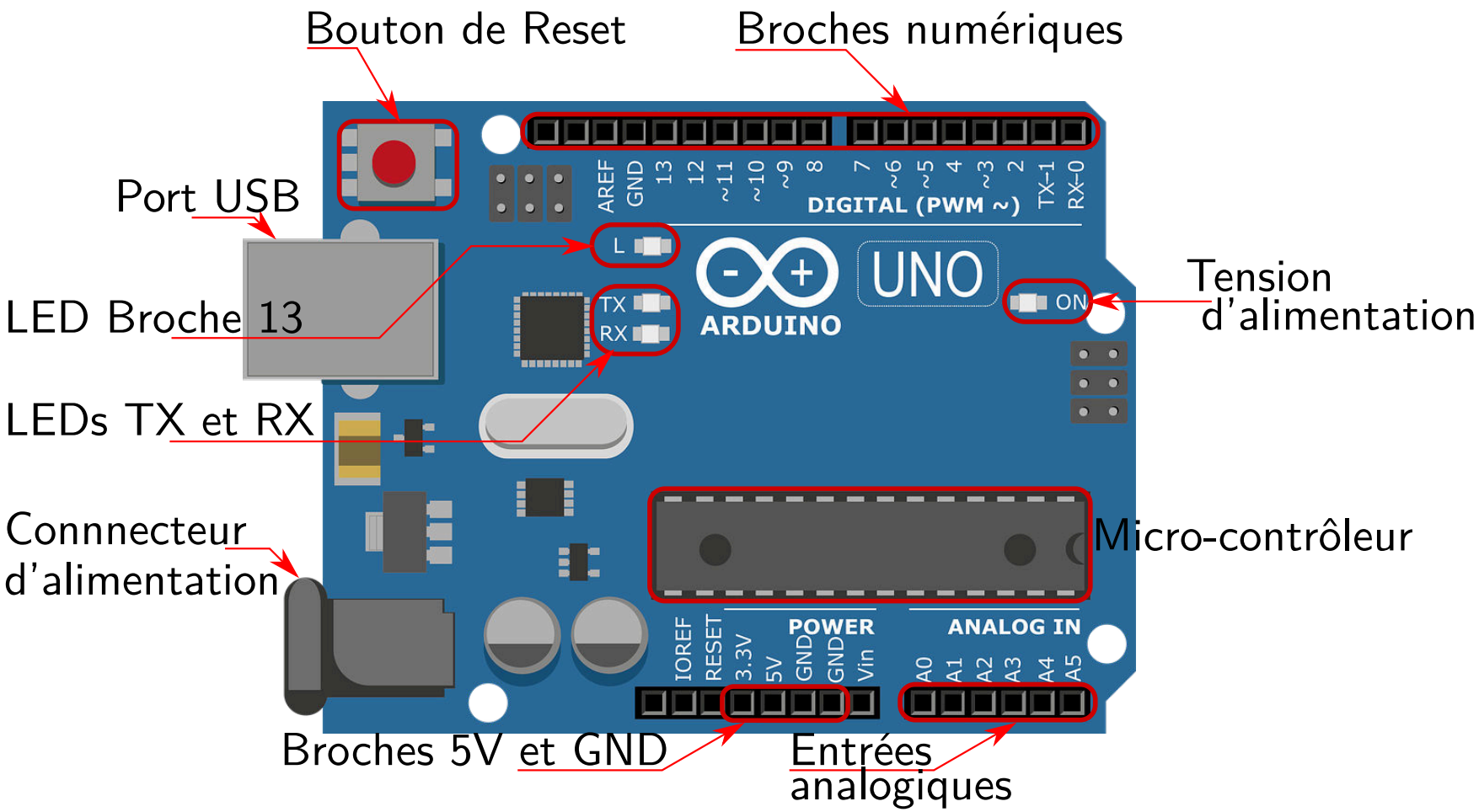
Analyse détaillée de la carte Arduino Uno



Microcontrôleur ATMEGA : cœur de votre carte Arduino.

Utilisation de la carte Arduino UNO

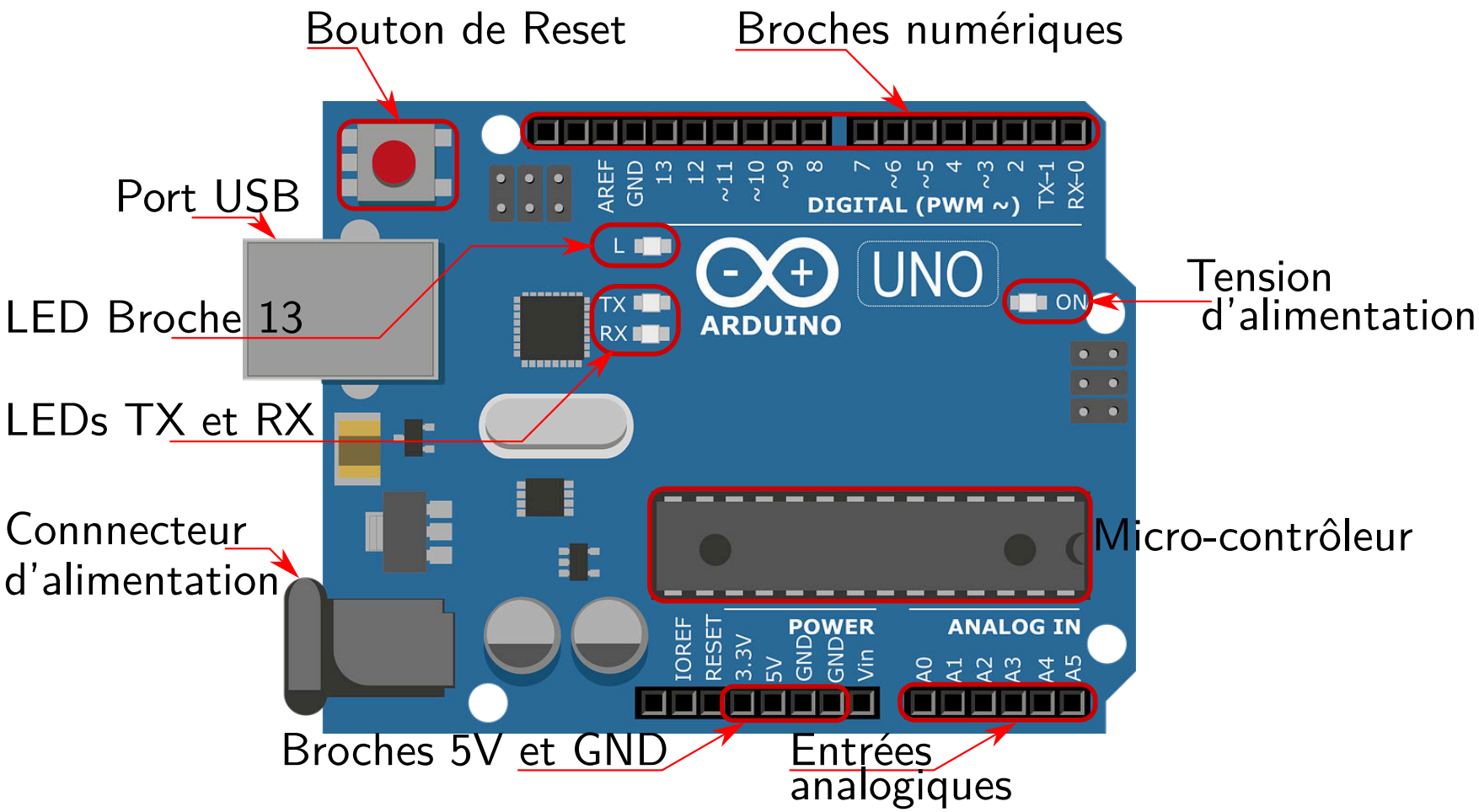
Analyse détaillée de la carte Arduino Uno



Entrées analogiques : entrées analogiques sur 10 bits. Le code utilisable est : `analogRead()`.

Utilisation de la carte Arduino UNO

Analyse détaillée de la carte Arduino Uno



Broches 5 V et GND : possibilités d'alimentation de votre circuit via la carte Arduino.

La communication avec le PC peut être décrite comme suit :

- quatorze entrées – sorties logiques (Pin Digital de 0 à 13) :
  - série asynchrone (0 : Rx et 1 : Tx) (les pins 0 et 1 ne seront donc pas utilisables) ;
  - deux interruptions externes sur 2 et 3 (utilisées pour le codeur en quadrature) ;
  - sortie 13 couplée à une LED sur la carte ;
  - parmi ces 14 entrées/sorties, six d'entre elles peuvent être utilisées comme des sorties analogiques PWM (3,5,6,9,10 et 11).
- six entrées analogiques :
  - tension d'entrée inférieure à la tension de référence ;
  - six CAN 10 bits (plage de 1024) ;
  - peuvent aussi fonctionner comme des E/S numériques.

Utilisation de la carte Arduino UNO

Mise en place de la programmation

Compilation et vérification du programme

Envoie du programme à la carte Arduin

Programme

pinMode(pin,mode)

Association d'une variable déclarée précédemment à une fonction de type `Sortie` ou `Entrée`.

```
int pin = 7;
void setup()
{
// Association de la connection digitale àla sortie
pinMode(pin,OUTPUT);
}
void loop()
{
}
```

digitalRead(pin)

Lecture d'une entrée déclarée et renvoie d'une valeur binaire HIGH ou LOW.

```
int pin = 7;
int val= 0 ;
void setup()
{
// Association de la connection digitale àune entrée
pinMode(pin,INPUT);
}
void loop()
{
// Lecture de l'entrée définie
val = digitalRead(pin);
}
```

digitalWrite(pin,value)

Association d'une valeur binaire HIGH ou LOW à la sortie appelée pin.

```
int pin = 13;
void setup()
{
// Association de la connection digitale àune sortie
pinMode(Pin,OUTPUT);
}
void loop()
{
// Passer le pin utilisé àla valeur haute
digitalWrite(pin,HIGH);
}
```

analogRead(pin)

Lecture d'une entrée obtenue à l'aide d'une tension comprise entre 0 et 5 Volts et renvoie d'une valeur entière `int(0 to 1023)`.

```
// un potentiomètre raccordé par exemple
int pin = 3;
int val= 0 ;
void setup()
{
}
void loop()
{
// Lecture de l'entrée analogique
val=analogRead(pin);
}
```

analogWrite(pin,value)

Associe une entrée analogique à un pin en passant par un branchement PWM.

```
// Connection d une LED
int pin=9;
// Un potentiomètre raccordé par exemple
int pin2=3;
int val= 0 ;
void setup()
{
pinMode(pin,OUTPUT);
}
void loop()
{
// Lecture de l'entrée analogique (entre 0 et 1023)
val=analogRead(pin2);
// Envoie de la valeur analogique divisée par 4 àla LED (0 à255)
analogWrite(pin,val/4);
}
```

`delay(ms)`

Ajout d'un temps de pause (supérieur à celui de l'itération classique) exprimé en millisecondes.

`delayMicroseconds(`μ`s)`

Ajout d'un temps de pause (supérieur à celui de l'itération classique) exprimé en microsecondes.

`delay(ms)`

Ajout d'un temps de pause (supérieur à celui de l'itération classique) exprimé en millisecondes.

`delayMicroseconds(`μ`s)`

Ajout d'un temps de pause (supérieur à celui de l'itération classique) exprimé en microsecondes.

Affichage des sorties 1/2

Afin de pouvoir consulter les sorties, il faut paramétrer la communication :

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

- cela fixe le débit de communication en nombre de caractères par secondes (unité : baud) pour la communication série ;
- pour communiquer avec l'ordinateur, il faut utiliser un de ces débits : 300, 1200, 2400, 4800 ... 115200.

Plus le débit est rapide, plus la communication est rapide.

Affichage des sorties 2/2

Une fois que la communication est paramétrée, il est possible d'afficher les sorties analogiques ou numériques.

```
void setup()
{
Serial.begin(9600);
}
void loop()
{
Serial.print(val);
}
```

Pour afficher ces sorties, il suffit alors d'activer le moniteur et de le régler à la bonne fréquence.

La fonction `Serial.println(val)` inclut un retour à la ligne.

1 2 3 4 5 6 7

Prise en main de la carte Arduino UNO

Prise en main de la carte Arduino UNO

Exemple 1 : Allumage d'une LED manuellement

Description de l'application :

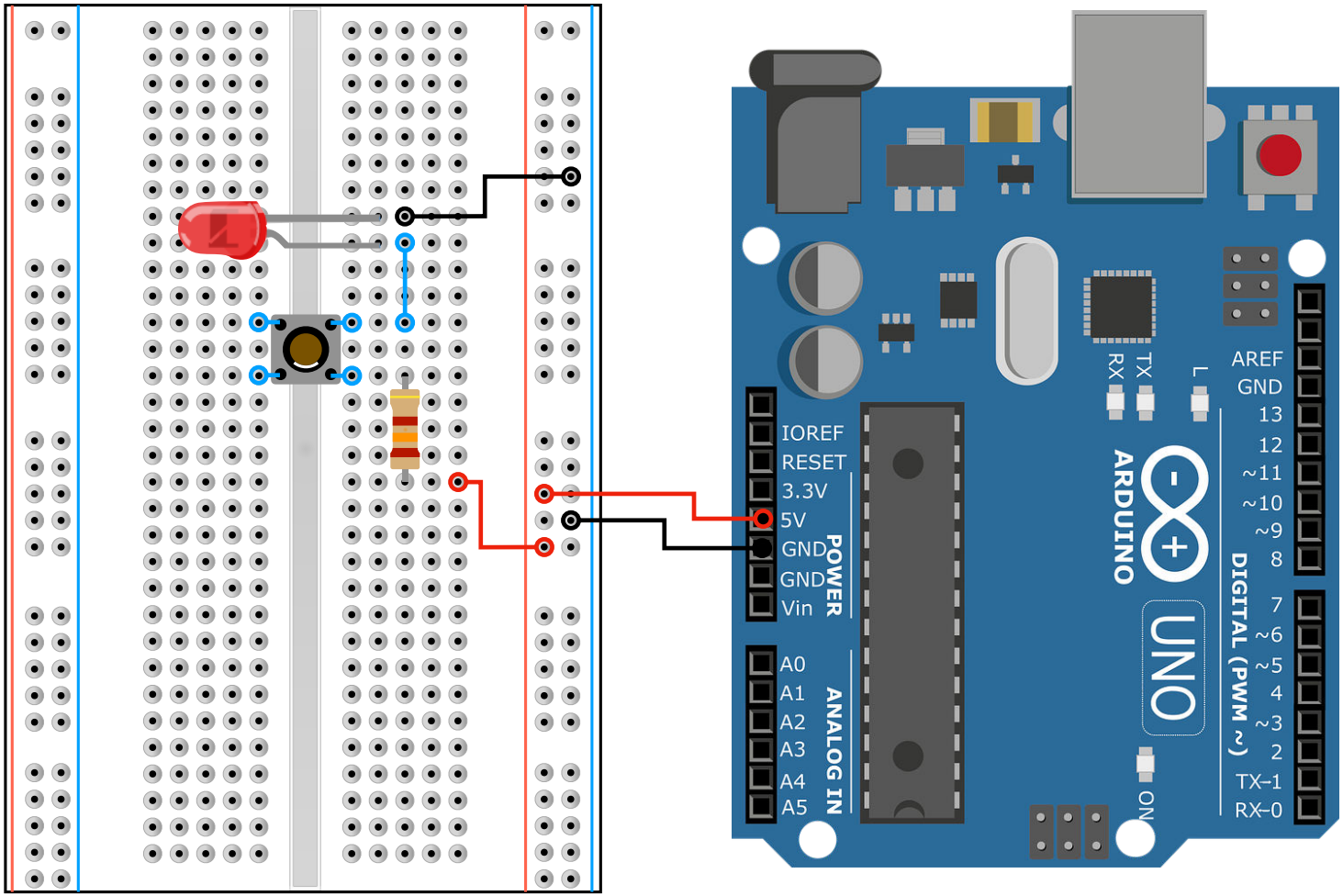
- Vérifier que votre carte Arduino est **débranchée** ;
- connecter un fil rouge aux 5V de l'Arduino et mettez l'autre côté dans un bus d'alimentation de votre breadbord ;
- faites de même (sur un autre bus) avec la masse (GND) de la carte Arduino ;
- placer le bouton poussoir au-dessus de la ligne centrale de votre breadboard (le montage est possible dans un seul sens donc il est inutile de forcer !) ;
- placer une résistance de 230 Ω en entrée du bouton poussoir ;
- ajouter une LED en sortie du bouton poussoir en connectant la cathode à la masse ;
- brancher le câble USB afin d'obtenir une alimentation.

Objectif

Comprendre la notion de création de circuits avec pilotage manuel.

Prise en main de la carte Arduino UNO

Exemple 1 : Allumage d'une LED manuellement



Prise en main de la carte Arduino UNO

Exemple 2 : Allumage d'une LED programmé

Description de l'application :

- réaliser le schéma suivant avec une LED, une résistance de 230W entre la masse GND et la sortie 2 ;
- faire valider le câblage par le professeur ;
- télécharger sur le site, le dossier zippé Arduino_decouverte.zip contenant l'ensemble des programmes utiles pour cette initiation ;
- dézipper le dossier ;
- lancer le logiciel Arduino sur le bureau et ouvrir le projet LED.ino.

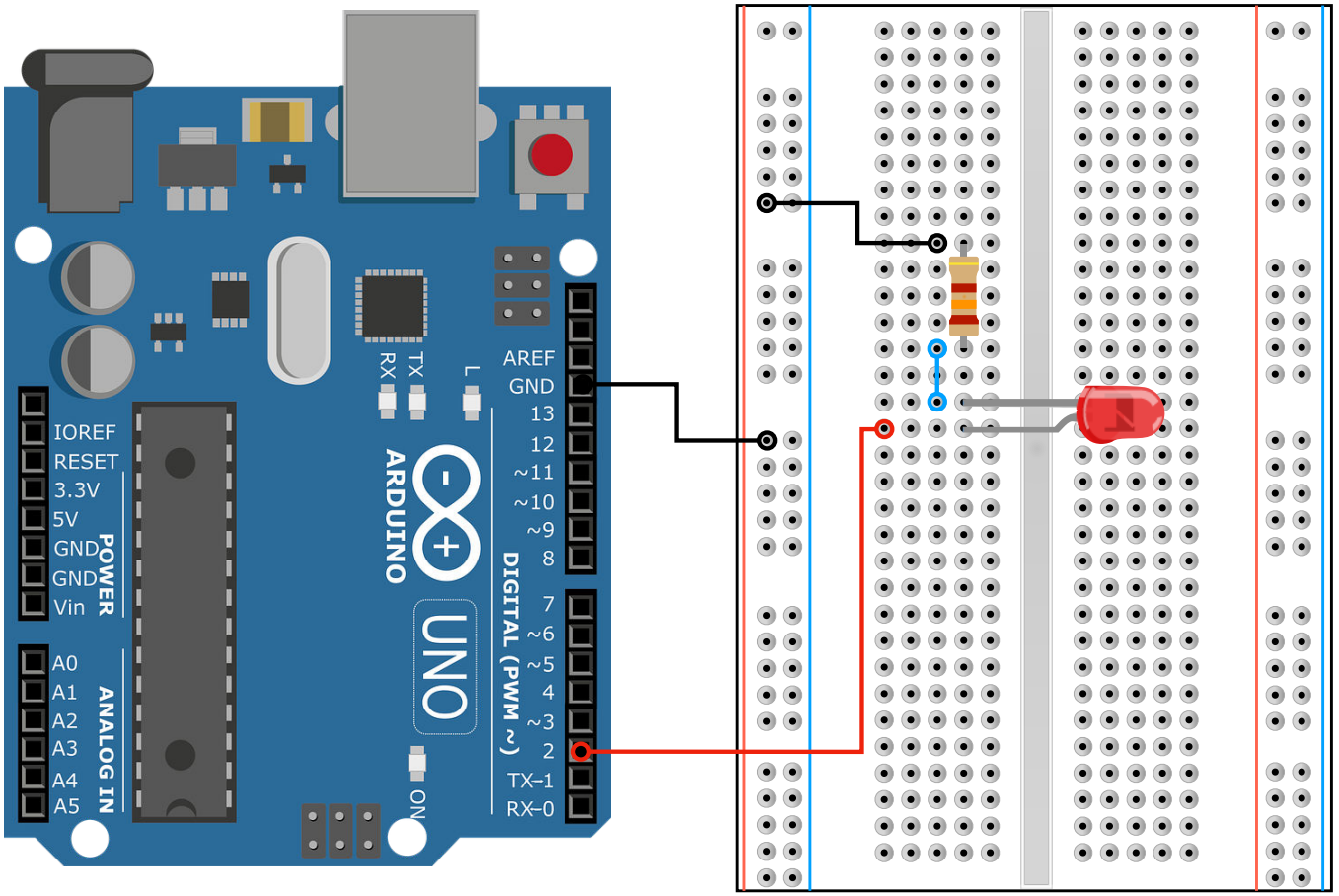
Objectif

Comprendre la notion de programmation utile pour faire clignoter une LED.

Prise en main de la carte Arduino UNO

Exemple 2 : Allumage d'une LED programmé

Branchement à réaliser



Prise en main de la carte Arduino UNO

Exemple 2 : Allumage d'une LED programmé

Description du programme

- `int var = val` : association d'une valeur à une variable ;
- `pinMode(pin,mode)` : associer le rôle à la valeur choisie ;
- `digitalWrite(pin,value)` : si le `pinMode` a été configuré tel un `OUTPUT` avec `pinMode()`, sa tension correspondra à son positionnement (value) : 5V pour `HIGH`, 0V for `LOW` ;
- `delay(ms)` : configuration d'une pause dans le programme avec le temps souhaité ici en ms (`delayMicroseconds(`μ`s)`).

Description du programme

- `int var = val` : association d'une valeur à une variable ;
- `pinMode(pin,mode)` : associer le rôle à la valeur choisie ;
- `digitalWrite(pin,value)` : si le `pinMode` a été configuré tel un `OUTPUT` avec `pinMode()`, sa tension correspondra à son positionnement (value) : 5V pour `HIGH`, 0V for `LOW` ;
- `delay(ms)` : configuration d'une pause dans le programme avec le temps souhaité ici en ms (`delayMicroseconds(µs)`).

Prise en main de la carte Arduino UNO

Exemple 2 : Allumage d'une LED programmé

Description du programme

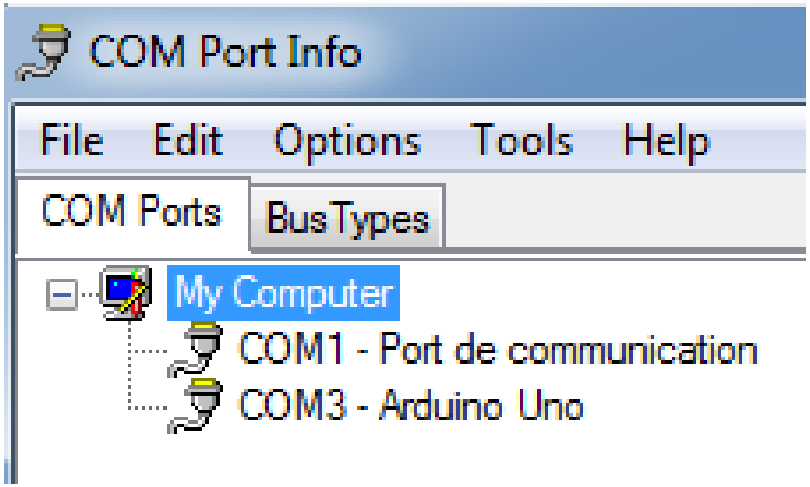
- `int var = val` : association d'une valeur à une variable ;
- `pinMode(pin,mode)` : associer le rôle à la valeur choisie ;
- `digitalWrite(pin,value)` : si le `pinMode` a été configuré tel un `OUTPUT` avec `pinMode()`, sa tension correspondra à son positionnement (value) : 5V pour `HIGH`, 0V for `LOW` ;
- `delay(ms)` : configuration d'une pause dans le programme avec le temps souhaité ici en ms (`delayMicroseconds(`μ`s)`).

Description du programme

- `int var = val` : association d'une valeur à une variable ;
- `pinMode(pin,mode)` : associer le rôle à la valeur choisie ;
- `digitalWrite(pin,value)` : si le `pinMode` a été configuré tel un `OUTPUT` avec `pinMode()`, sa tension correspondra à son positionnement (value) : 5V pour `HIGH`, 0V for `LOW` ;
- `delay(ms)` : configuration d'une pause dans le programme avec le temps souhaité ici en ms (`delayMicroseconds(`μ`s)`).

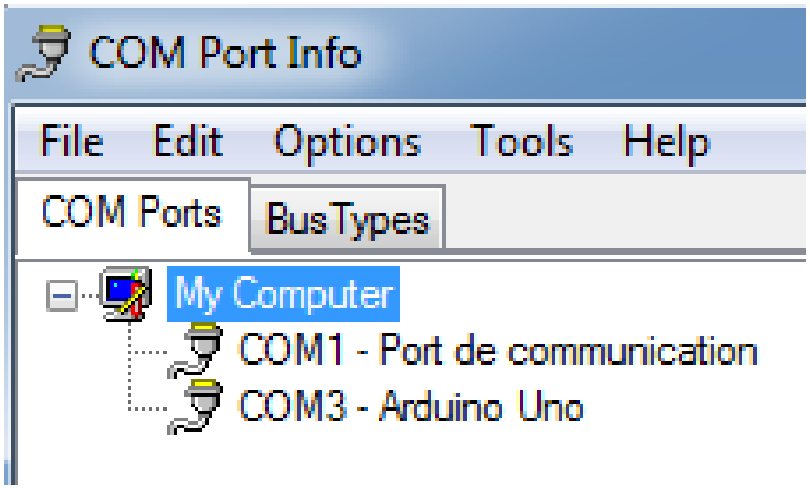
Test du programme

- Brancher la liaison USB entre le PC et la carte Arduino Uno.
- Ouvrir le gestionnaire de périphérique dans le panneau de configuration et vérifier la présence du matériel dans Ports (COM et LPT).
- Retenir le port COM utilisé. Le numéro port COM doit être inférieur à 9. Si ce n'est pas le cas, il faut le changer dans les paramètres avancés.
- Sélectionner dans Outils/Port Série quel com est utilisé par la carte Arduino.



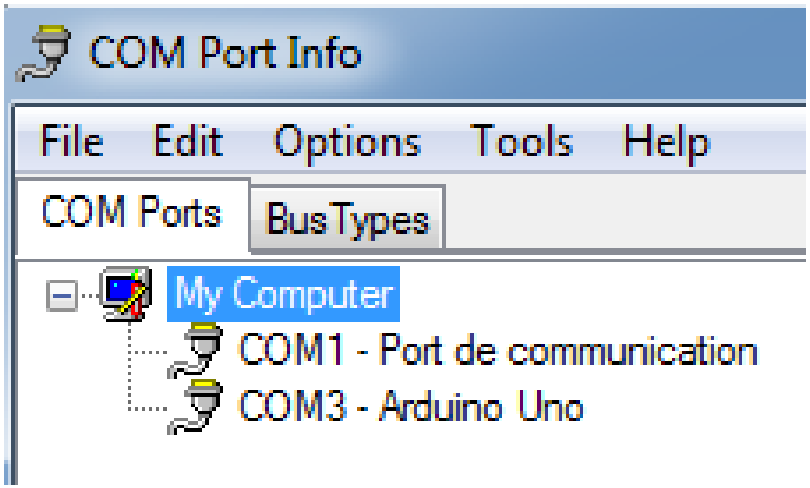
Test du programme

- Brancher la liaison USB entre le PC et la carte Arduino Uno.
- Ouvrir le gestionnaire de périphérique dans le panneau de configuration et vérifier la présence du matériel dans Ports (COM et LPT).
- Retenir le port COM utilisé. Le numéro port COM doit être inférieur à 9. Si ce n'est pas le cas, il faut le changer dans les paramètres avancés.
- Sélectionner dans Outils/Port Série quel com est utilisé par la carte Arduino.



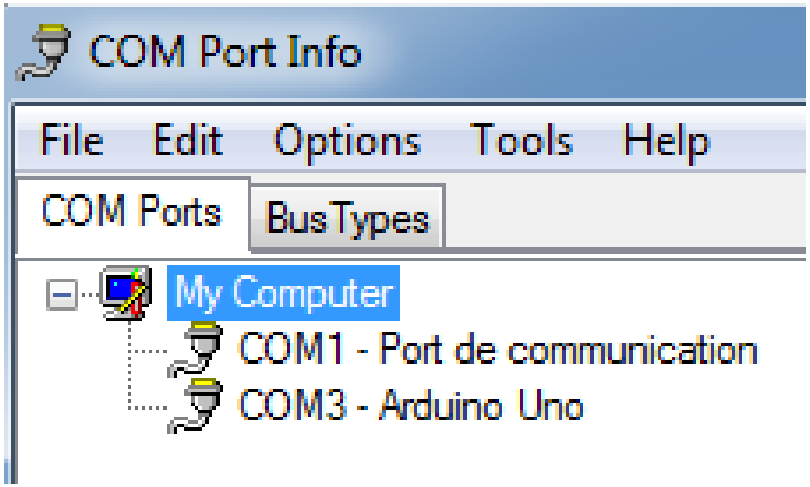
Test du programme

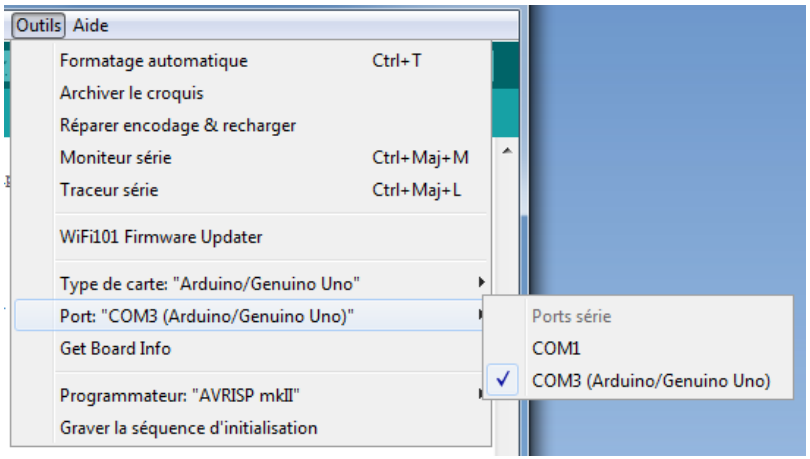
- Brancher la liaison USB entre le PC et la carte Arduino Uno.
- Ouvrir le gestionnaire de périphérique dans le panneau de configuration et vérifier la présence du matériel dans Ports (COM et LPT).
- Retenir le port COM utilisé. Le numéro port COM doit être inférieur à 9. Si ce n'est pas le cas, il faut le changer dans les paramètres avancés.
- Sélectionner dans Outils/Port Série quel com est utilisé par la carte Arduino.



Test du programme

- Brancher la liaison USB entre le PC et la carte Arduino Uno.
- Ouvrir le gestionnaire de périphérique dans le panneau de configuration et vérifier la présence du matériel dans Ports (COM et LPT).
- Retenir le port COM utilisé. Le numéro port COM doit être inférieur à 9. Si ce n'est pas le cas, il faut le changer dans les paramètres avancés.
- Sélectionner dans Outils/Port Série quel com est utilisé par la carte Arduino.



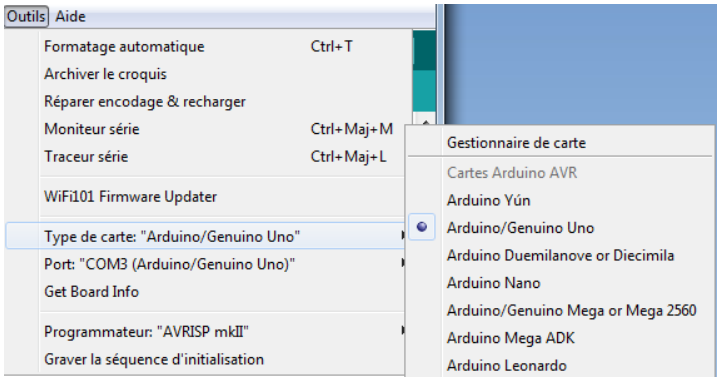


Prise en main de la carte Arduino UNO

Exemple 2 : Allumage d'une LED programmé

Test du programme

- Vérifier le type de carte utilisé.
- Compiler et vérifier le programme.
- Transférer le programme dans la carte.
- Constater que la LED clignote.



Question

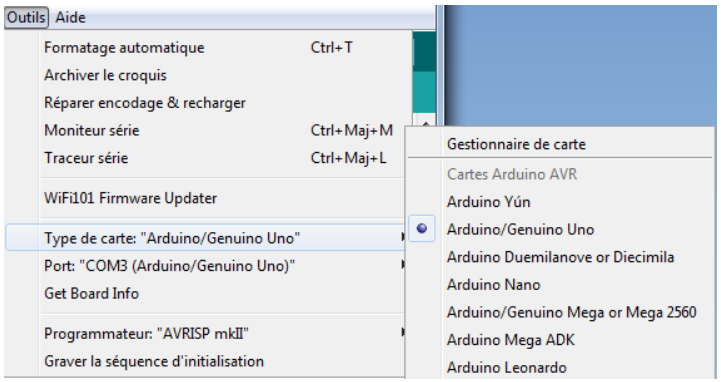
Proposer une programmation permettant, en rajoutant une LED sur une sortie digitale de la broche numérique, d'avoir un clignotement périodique des deux LED.

Prise en main de la carte Arduino UNO

Exemple 2 : Allumage d'une LED programmé

Test du programme

- Vérifier le type de carte utilisé.
- Compiler et vérifier le programme.
- Transférer le programme dans la carte.
- Constater que la LED clignote.



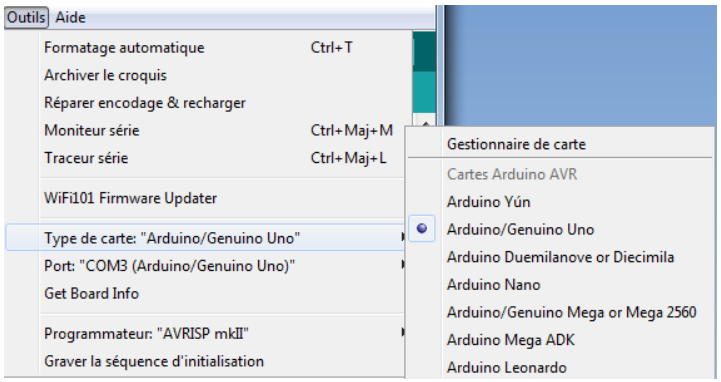
Question

Proposer une programmation permettant, en rajoutant une LED sur une sortie digitale de la broche numérique, d'avoir un clignotement périodique des deux LED.

Vérification du programme

Test du programme

- Vérifier le type de carte utilisé.
- Compiler et vérifier le programme.
- Transférer le programme dans la carte.
- Constater que la LED clignote.



Question

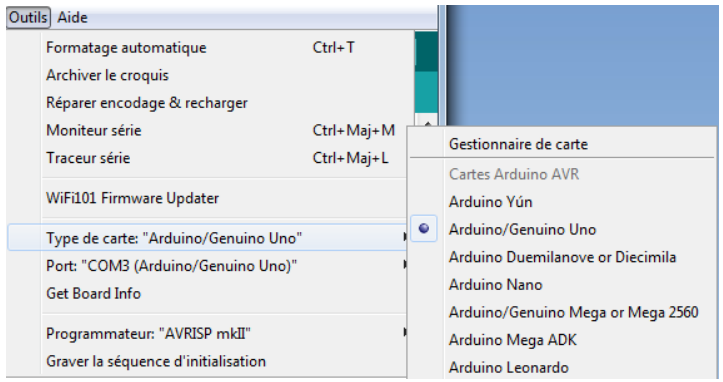
Proposer une programmation permettant, en rajoutant une LED sur une sortie digitale de la broche numérique, d'avoir un clignotement périodique des deux LED.

Vérification du programme

Transfert du programme dans la carte Arduino

Test du programme

- Vérifier le type de carte utilisé.
- Compiler et vérifier le programme.
- Transférer le programme dans la carte.
- Constater que la LED clignote.



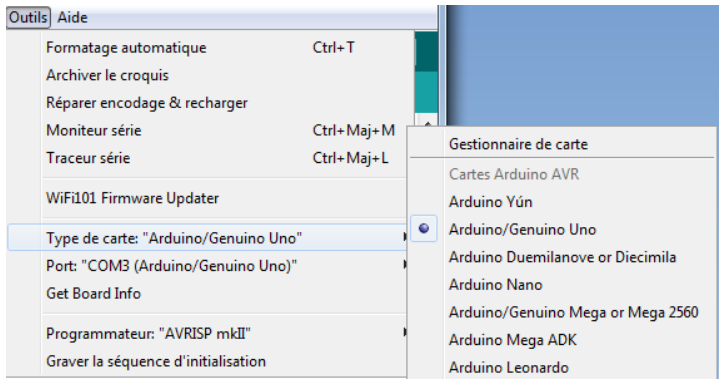
Question

Proposer une programmation permettant, en rajoutant une LED sur une sortie digitale de la broche numérique, d'avoir un clignotement périodique des deux LED.

Vérification du programme

Transfert du programme dans la carte Arduino

Test du programme



- Vérifier le type de carte utilisé.
- Compiler et vérifier le programme.
- Transférer le programme dans la carte.
- Constater que la LED clignote.

Question

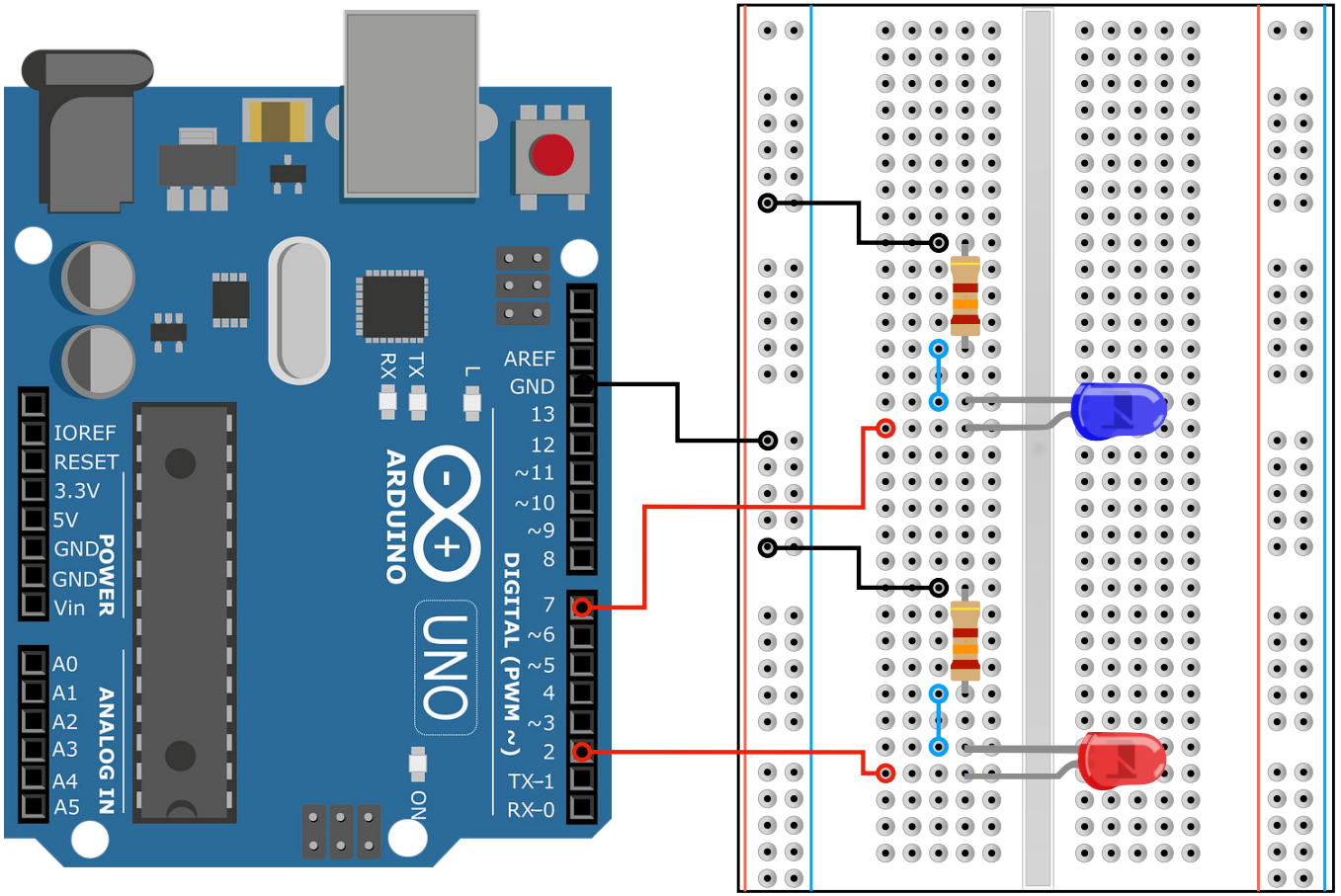
Proposer une programmation permettant, en rajoutant une LED sur une sortie digitale de la broche numérique, d'avoir un clignotement périodique des deux LED.

Vérification du programme

Transfert du programme dans la carte Arduino

Prise en main de la carte Arduino UNO

Exemple 2 : Allumage d'une LED programmé



Prise en main de la carte Arduino UNO

Exemple 2 : Allumage d'une LED programmé

Dans ce cas, la zone de boucle infinie est utilisée afin de maintenir l'allumage et l'extinction des deux LEDs utilisées.

Et là... c'est le drame ! En effet, tout est ok, Arduino exécute la première instruction une fois dans le setup, puis la loop exécute le reste à l'infini : j'allume, j'éteins, j'allume, j'éteins... Et pourtant la diode reste allumée !

Voilà la situation la plus délicate pour un programmeur : il n'y a pas de bug, tout fonctionne et pourtant le résultat n'est pas là. Alors que faire ?

Prise en main de la carte Arduino UNO

Exemple 2 : Allumage d'une LED programmé

Dans ce cas, la zone de boucle infinie est utilisée afin de maintenir l'allumage et l'extinction des deux LEDs utilisées.
Et là... c'est le drame ! En effet, tout est ok, Arduino exécute la première instruction une fois dans le `setup`, puis la loop exécute le reste à l'infini : j'allume, j'éteins, j'allume, j'éteins... Et pourtant la diode reste allumée !

Voilà la situation la plus délicate pour un programmeur : il n'y a pas de bug, tout fonctionne et pourtant le résultat n'est pas là. Alors que faire ?

Prise en main de la carte Arduino UNO

Exemple 2 : Allumage d'une LED programmé

Dans ce cas, la zone de boucle infinie est utilisée afin de maintenir l'allumage et l'extinction des deux LEDs utilisées.
Et là... c'est le drame ! En effet, tout est ok, Arduino exécute la première instruction une fois dans le `setup`, puis la loop exécute le reste à l'infini : j'allume, j'éteins, j'allume, j'éteins... Et pourtant la diode reste allumée !

Voilà la situation la plus délicate pour un programmeur : il n'y a pas de bug, tout fonctionne et pourtant le résultat n'est pas là. Alors que faire ?

Notion de fréquence

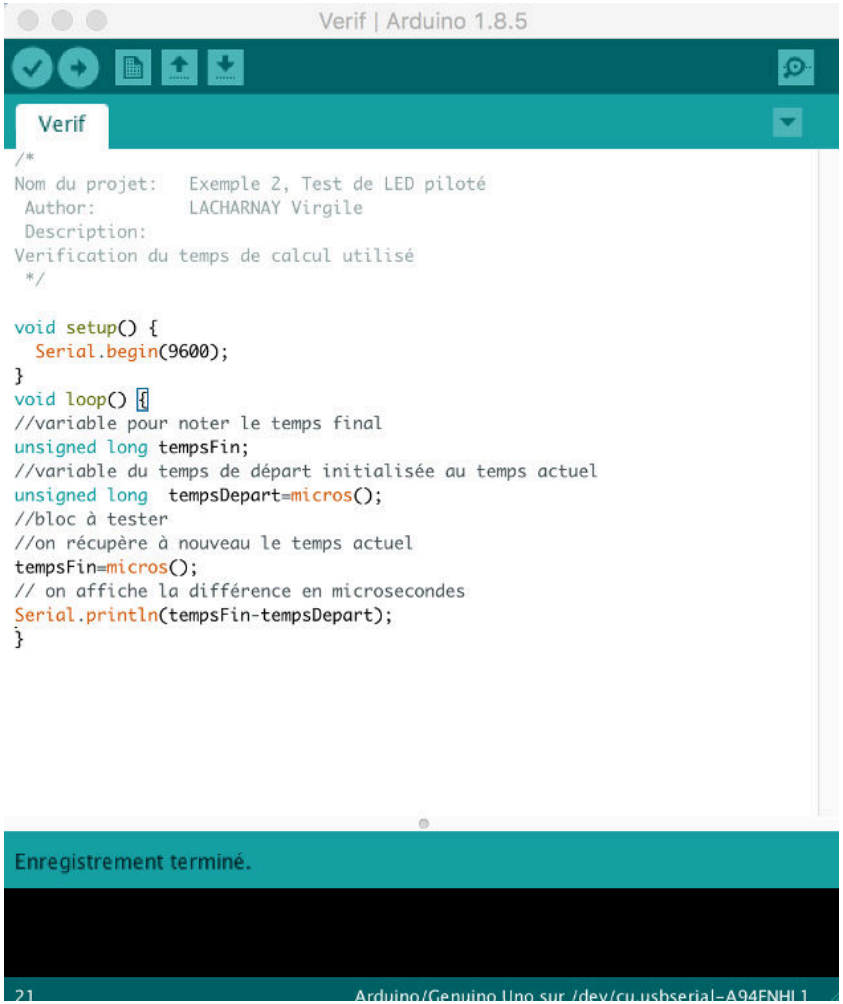
La fréquence représente le nombre de fois par seconde qu'une action est faite. Par exemple si on tape 10 touches par seconde sur mon clavier, la fréquence de frappe est de 10 hertz, on note 10 Hz.

Exemple	Fréquence	Action
Respiration	0,5 Hz	1 respiration toutes les 2 secondes
Seconde	1 Hz	1 seconde toutes les secondes
Pouls humain	1,2 Hz	1,2 battement par seconde
Courant alternatif	50 Hz	50 oscillations par seconde
La du diapason	440 Hz	440 oscillations par seconde
Ondes FM	entre 23 et 95 KHz	entre 23 000 et 95 000 vibrations par seconde
Arduino	16 MHz	soit 16 000 000 d'actions machine par seconde
Processeur actuel	3 Ghz	soit 3 000 000 000 d'actions machine par seconde

16 000 000 d'actions par seconde ne veut pas dire 16 000 000 d'instructions par seconde. Une action machine est un passage ou non d'un courant électrique (1 et 0). Une instruction nécessite une multitude d'actions machine pour se réaliser.

Prise en main de la carte Arduino UNO

Exemple 2 : Allumage d'une LED programmé



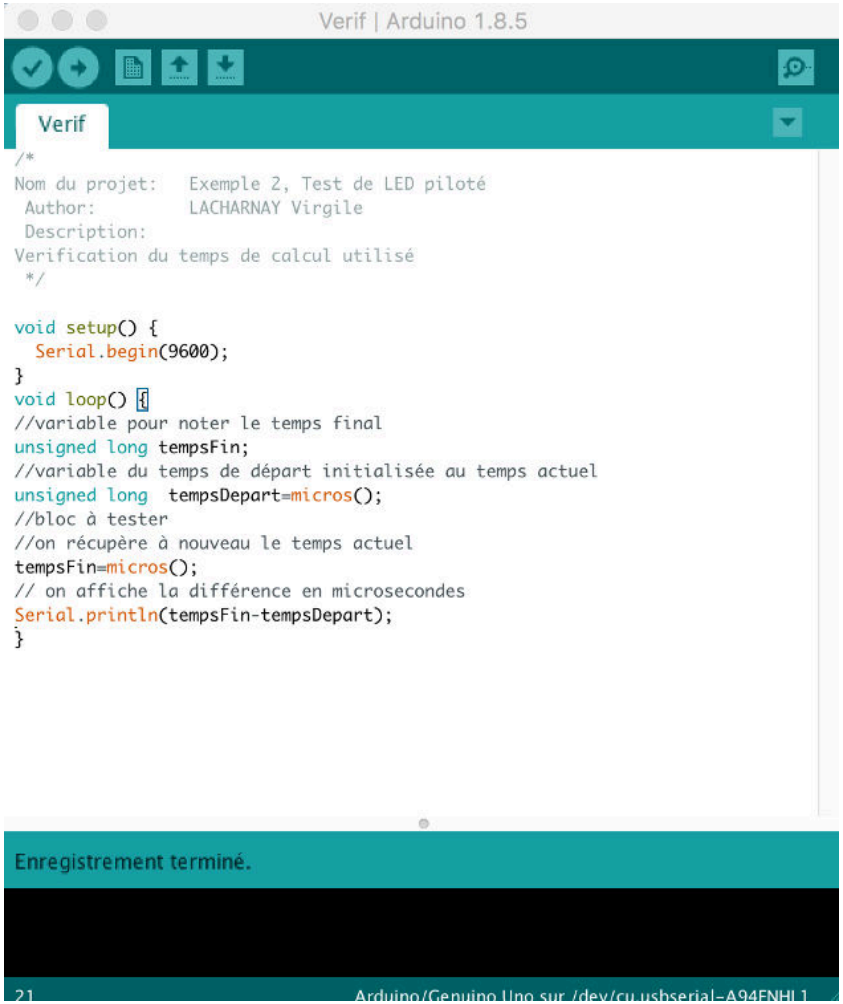
Utilisation de l'horloge interne de l'Arduino. Il est possible de stocker dans une variable le temps actuel et d'exécuter l'instruction (ou le bloc d'instructions) dont on veut mesurer la vitesse d'exécution, et on fait la différence entre le temps actuel et le temps stocké et on l'affiche sur la console.

Obligation d'utiliser une nouvelle fonction afin de laisser à la LED la possibilité de s'éteindre et de s'allumer de manière visible pour l'œil humain :

```
delay(ms)
```

Prise en main de la carte Arduino UNO

Exemple 2 : Allumage d'une LED programmé



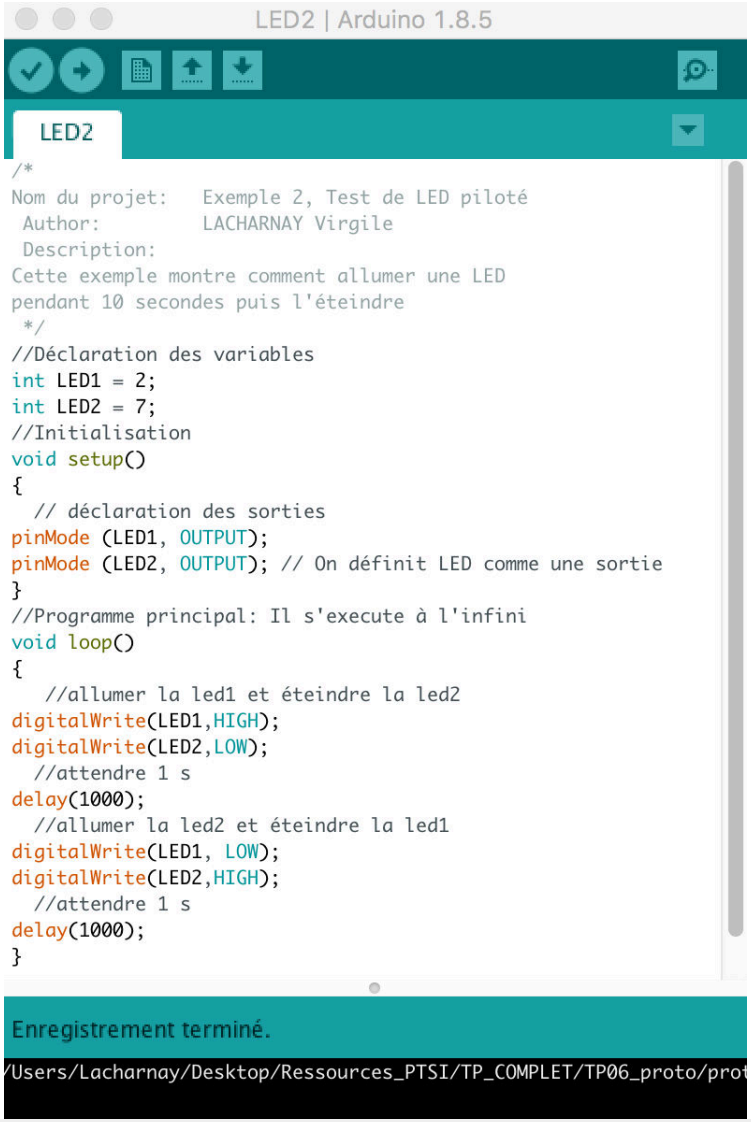
Utilisation de l'horloge interne de l'Arduino. Il est possible de stocker dans une variable le temps actuel et d'exécuter l'instruction (ou le bloc d'instructions) dont on veut mesurer la vitesse d'exécution, et on fait la différence entre le temps actuel et le temps stocké et on l'affiche sur la console.

Obligation d'utiliser une nouvelle fonction afin de laisser à la LED la possibilité de s'éteindre et de s'allumer de manière visible pour l'œil humain :

```
delay(ms)
```

Prise en main de la carte Arduino UNO

Exemple 2 : Allumage d'une LED programmé



Dans ce cas, la zone de boucle infinie est utilisée afin de maintenir l'allumage et l'extinction des deux LEDs utilisées. Il serait possible, dans la déclaration de variables de définir la période :

```
int T=1000;
```

Cela paraît plus judicieux, pour un programme plus lourd de n'utiliser dans la partie `void` et `loop` que des éléments déclarés.

Prise en main de la carte Arduino UNO

Exemple 3 : Variation de l'intensité lumineuse fournie par la LED

Description de l'application :

- réaliser le schéma suivant avec une LED rouge, une résistance de 230W entre la masse GND et la sortie 6, ainsi qu'un potentiomètre sur l'entrée analogique 2 ;
- faire valider le câblage par le professeur ;
- ouvrir le projet LED2.ino.

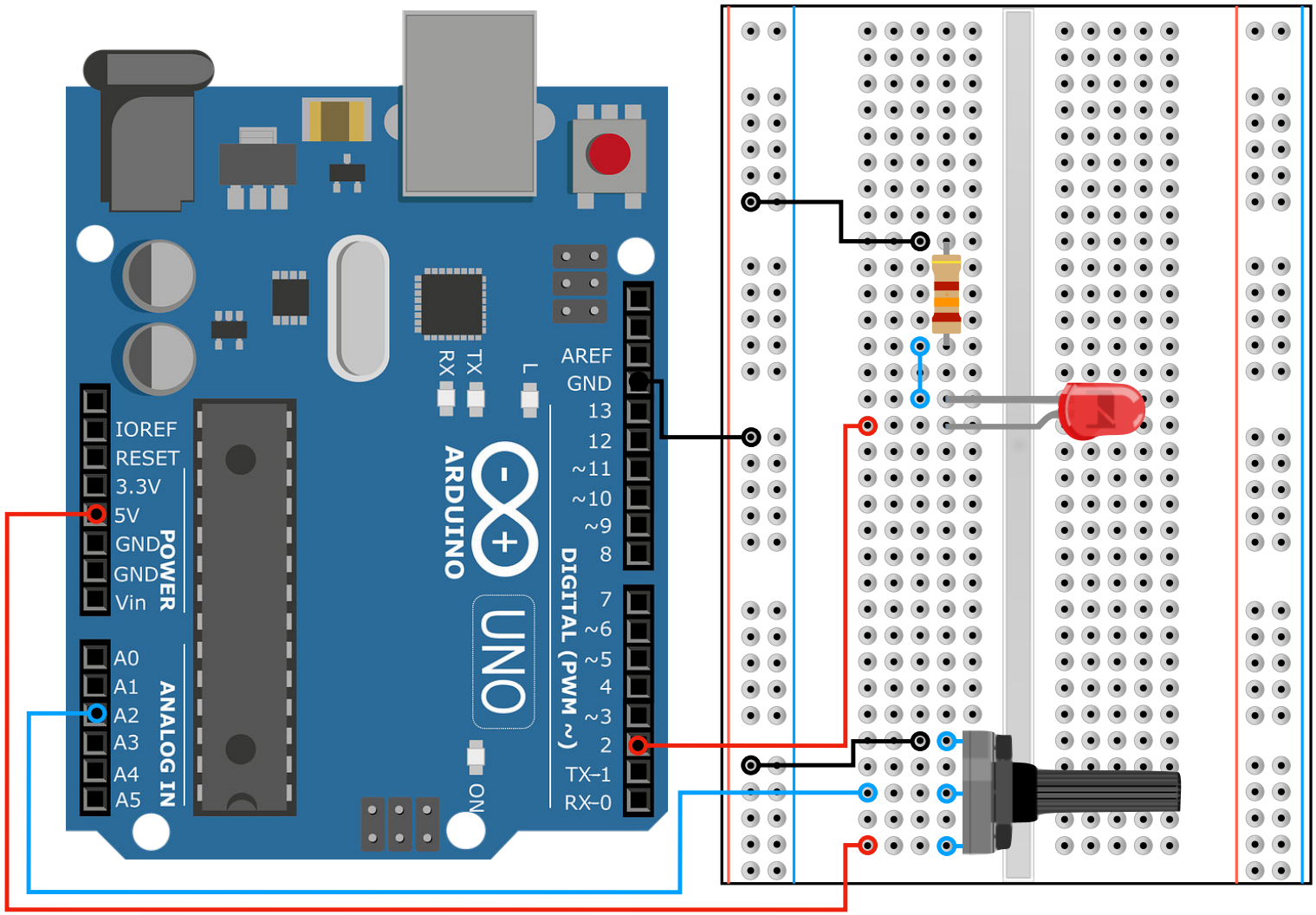
Objectif

Comprendre la notion de programmation utile pour faire varier l'intensité lumineuse de la LED en envoyant une tension modulée à partir de la lecture de la tension d'un potentiomètre.

Prise en main de la carte Arduino UNO

Exemple 3 : Variation de l'intensité lumineuse fournie par la LED

Branchement à réaliser



Prise en main de la carte Arduino UNO

Exemple 3 : Variation de l'intensité lumineuse fournie par la LED

Description du programme

- `const` : cette option empêche d'associer une autre valeur que celle fournie par l'entrée analogique 2 dans notre cas ;
- `Serial.begin()` : détail la communication utilisée avec le port de communication ;
- `analogRead(pin)` : signifie que le microprocesseur va lire la tension analogique sur l'entrée (pin) et va la coder sur 10 bits ;
- `analogWrite(pin,value)` : signifie que le microprocesseur va moduler la tension en sortie sur la sortie (pin) avec une valeur (value).

Prise en main de la carte Arduino UNO

Exemple 3 : Variation de l'intensité lumineuse fournie par la LED

Description du programme

- `const` : cette option empêche d'associer une autre valeur que celle fournie par l'entrée analogique 2 dans notre cas ;
- `Serial.begin()` : détail la communication utilisée avec le port de communication ;
- `analogRead(pin)` : signifie que le microprocesseur va lire la tension analogique sur l'entrée (pin) et va la coder sur 10 bits ;
- `analogWrite(pin,value)` : signifie que le microprocesseur va moduler la tension en sortie sur la sortie (pin) avec une valeur (value).

Prise en main de la carte Arduino UNO

Exemple 3 : Variation de l'intensité lumineuse fournie par la LED

Description du programme

- `const` : cette option empêche d'associer une autre valeur que celle fournie par l'entrée analogique 2 dans notre cas ;
- `Serial.begin()` : détail la communication utilisée avec le port de communication ;
- `analogRead(pin)` : signifie que le microprocesseur va lire la tension analogique sur l'entrée (pin) et va la coder sur 10 bits ;
- `analogWrite(pin,value)` : signifie que le microprocesseur va moduler la tension en sortie sur la sortie (pin) avec une valeur (value).

Prise en main de la carte Arduino UNO

Exemple 3 : Variation de l'intensité lumineuse fournie par la LED

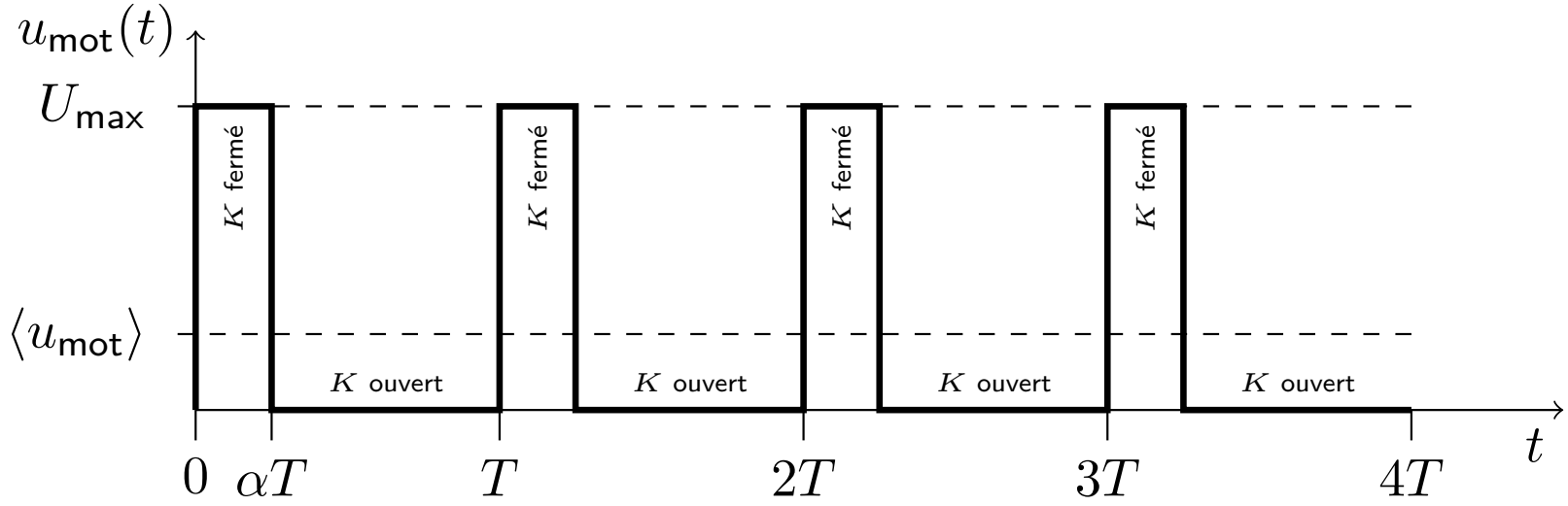
Description du programme

- `const` : cette option empêche d'associer une autre valeur que celle fournie par l'entrée analogique 2 dans notre cas ;
- `Serial.begin()` : détail la communication utilisée avec le port de communication ;
- `analogRead(pin)` : signifie que le microprocesseur va lire la tension analogique sur l'entrée (pin) et va la coder sur 10 bits ;
- `analogWrite(pin,value)` : signifie que le microprocesseur va moduler la tension en sortie sur la sortie (pin) avec une valeur (value).

Description de la variable largeur

Cette variable est codée sur 8 bits. Cela correspond à ce que nous verrons au cours de l'étude du CI 3 concernant l'analyse, la modélisation et le dimensionnement des chaînes de conversion électromécanique.

- Si largeur vaut 0, le rapport cyclique α vaut 0 et la tension moyenne 0.
- Si largeur vaut 123, le rapport cyclique α vaut $0,5$ et la tension moyenne $0,5 \cdot U_{\text{max}}$.
- Si largeur vaut 255, le rapport cyclique α vaut 1 et la tension moyenne U_{max}.



Prise en main de la carte Arduino UNO

Exemple 3 : Variation de l'intensité lumineuse fournie par la LED

Test du programme

- Brancher la liaison USB entre le PC et la carte Arduino Uno.
- Compiler et vérifier le programme.
- Transférer le programme dans la carte.
- Constater que la luminosité de la LED varie lorsque l'on tourne le potentiomètre.
- Lancer le moniteur série et visualiser les valeurs des grandeurs affichées dans la fenêtre du moniteur. Vous devrez régler la vitesse de transfert à 115200 bps (correspondant au nombre de caractères par seconde `Serial.begin()` utilisé dans la programmation Arduino).

Test du programme

- Brancher la liaison USB entre le PC et la carte Arduino Uno.
- Compiler et vérifier le programme.
- Transférer le programme dans la carte.
- Constater que la luminosité de la LED varie lorsque l'on tourne le potentiomètre.
- Lancer le moniteur série et visualiser les valeurs des grandeurs affichées dans la fenêtre du moniteur. Vous devrez régler la vitesse de transfert à 115200 bps (correspondant au nombre de caractères par seconde `Serial.begin()` utilisé dans la programmation Arduino).

Prise en main de la carte Arduino UNO

Exemple 3 : Variation de l'intensité lumineuse fournie par la LED

Test du programme

- Brancher la liaison USB entre le PC et la carte Arduino Uno.
- Compiler et vérifier le programme.
- Transférer le programme dans la carte.
- Constater que la luminosité de la LED varie lorsque l'on tourne le potentiomètre.
- Lancer le moniteur série et visualiser les valeurs des grandeurs affichées dans la fenêtre du moniteur. Vous devrez régler la vitesse de transfert à 115200 bps (correspondant au nombre de caractères par seconde `Serial.begin()` utilisé dans la programmation Arduino).

Prise en main de la carte Arduino UNO

Exemple 3 : Variation de l'intensité lumineuse fournie par la LED

Test du programme

- Brancher la liaison USB entre le PC et la carte Arduino Uno.
- Compiler et vérifier le programme.
- Transférer le programme dans la carte.
- Constater que la luminosité de la LED varie lorsque l'on tourne le potentiomètre.
- Lancer le moniteur série et visualiser les valeurs des grandeurs affichées dans la fenêtre du moniteur. Vous devrez régler la vitesse de transfert à 115200 bps (correspondant au nombre de caractères par seconde `Serial.begin()` utilisé dans la programmation Arduino).

Test du programme

- Brancher la liaison USB entre le PC et la carte Arduino Uno.
- Compiler et vérifier le programme.
- Transférer le programme dans la carte.
- Constater que la luminosité de la LED varie lorsque l'on tourne le potentiomètre.
- Lancer le moniteur série et visualiser les valeurs des grandeurs affichées dans la fenêtre du moniteur. Vous devrez régler la vitesse de transfert à 115200 bps (correspondant au nombre de caractères par seconde `Serial.begin()` utilisé dans la programmation Arduino).

1 2 3 4 5 6 7

Exercice : Programmation de feux rouges

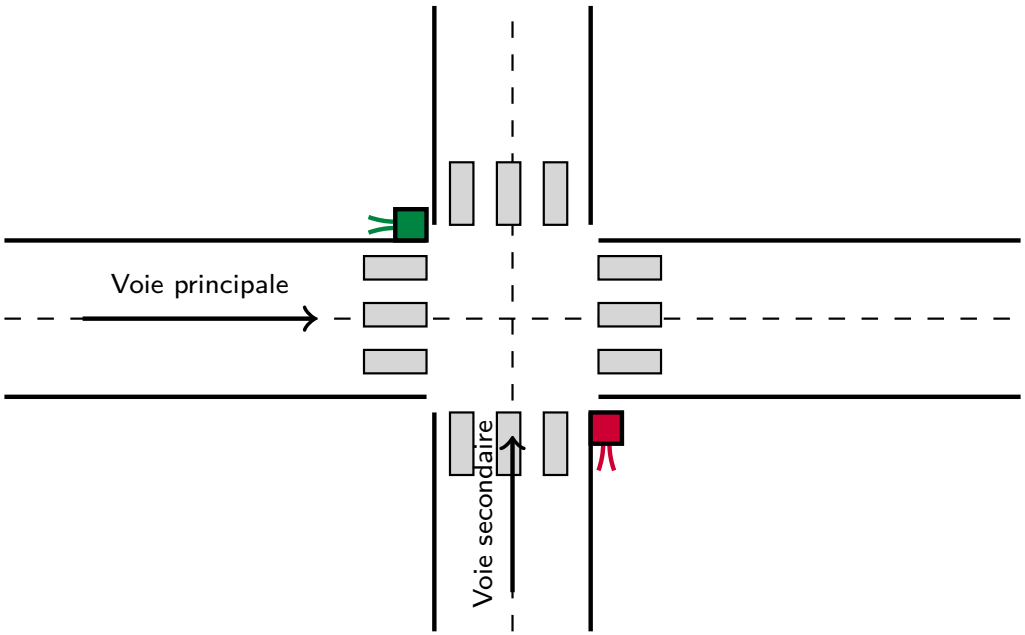
Exercice : Programmation de feux rouges

Première problématique

Objectif

Programmer un ensemble urbain avec une modélisation des feux de croisement sur carte Arduino.

On considère deux voies de circulation à sens unique avec un croisement comportant donc deux feux tricolores.



Exercice : Programmation de feux rouges

Première problématique

Problème 1

En journée, la programmation souhaitée est la suivante :

- le feu principal passe au vert durant 5 secondes ;
- il passe alors au orange pendant 2 secondes ;
- les deux feux restent au rouge durant 1 seconde ;
- le feu secondaire passe au vert durant 5 secondes ;
- il passe alors au orange pendant 2 secondes ;
- les deux feux restent au rouge durant 1 seconde ;

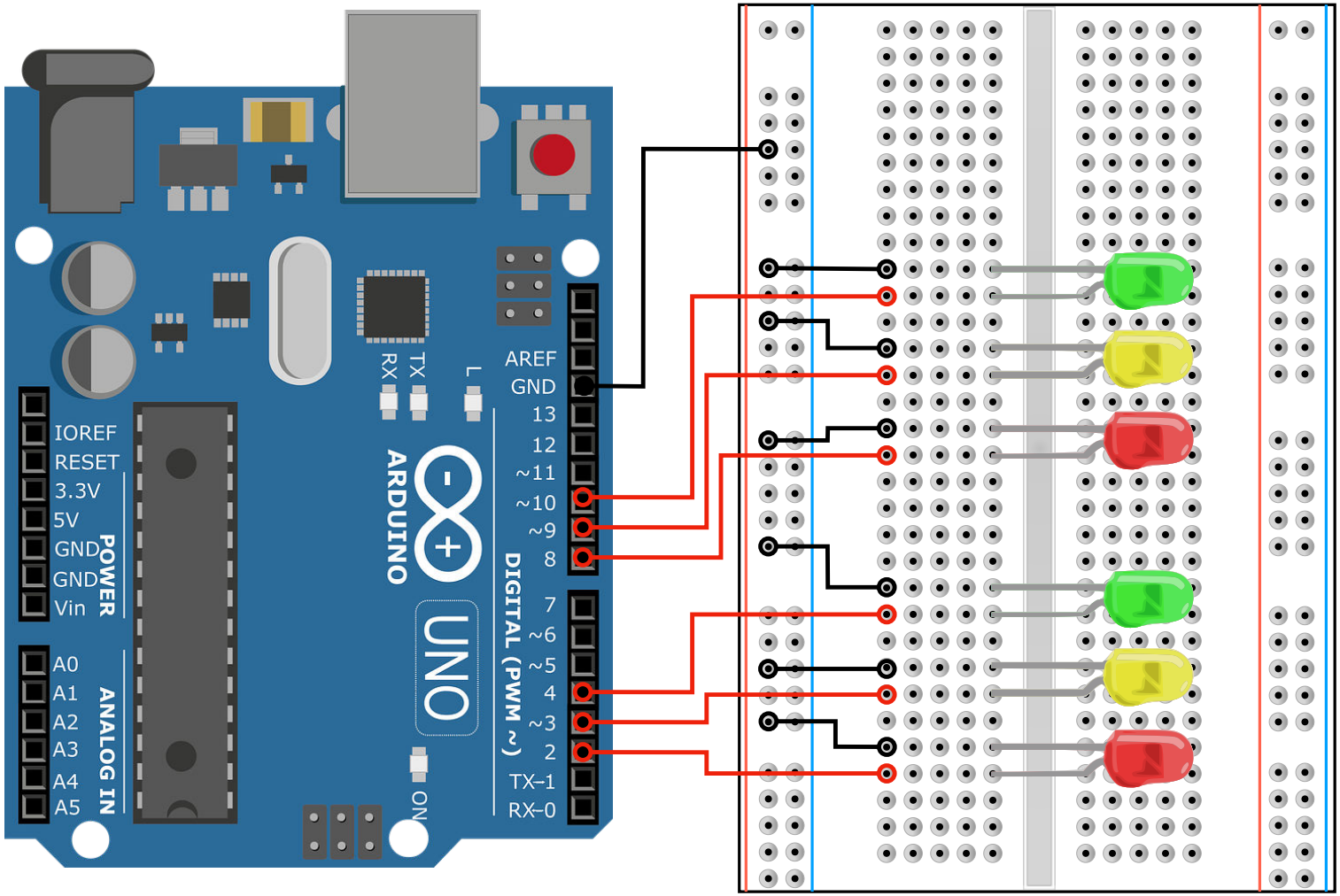
Il est nécessaire de rajouter des éléments de sécurité à cette description de fonctionnement basique.

Réaliser une programmation et un montage Arduino permettant de respecter cette coordination de feux de croisement.

Exercice : Programmation de feux rouges

Première problématique

Branchement à réaliser



Exercice : Programmation de feux rouges

Première problématique

Test du programme

```
// Définition des broches
const int led_rouge_feux_1 = 2;
const int led_jaune_feux_1 = 3;
const int led_verte_feux_1 = 4;
const int led_rouge_feux_2 = 8;
const int led_jaune_feux_2 = 9;
const int led_verte_feux_2 = 10;
```

Exercice : Programmation de feux rouges

Première problématique

Test du programme

```
void setup()
{
    // initialisation en sortie de toutes les broches
    pinMode(led_rouge_feux_1, OUTPUT);
    pinMode(led_jaune_feux_1, OUTPUT);
    pinMode(led_verte_feux_1, OUTPUT);
    pinMode(led_rouge_feux_2, OUTPUT);
    pinMode(led_jaune_feux_2, OUTPUT);
    pinMode(led_verte_feux_2, OUTPUT);

    // on initialise toutes les LED éteintes au début du programme
    // (sauf les deux feux rouges)
    digitalWrite(led_rouge_feux_1, HIGH);
    digitalWrite(led_jaune_feux_1, LOW);
    digitalWrite(led_verte_feux_1, LOW);
    digitalWrite(led_rouge_feux_2, HIGH);
    digitalWrite(led_jaune_feux_2, LOW);
    digitalWrite(led_verte_feux_2, LOW);
}
```

Exercice : Programmation de feux rouges
Première problématique

Test du programme

```
void loop()
{
    delay(1000);
// première séquence
    // 1 passe au vert
    digitalWrite(led_verte_feux_1, HIGH);
    digitalWrite(led_rouge_feux_1, LOW);
    // 2 reste au rouge
    delay(5000);
// deuxième séquence
    // 1 passe au orange
    digitalWrite(led_jaune_feux_1, HIGH);
    digitalWrite(led_verte_feux_1, LOW);
    // 2 reste rouge
    delay(2000);
 // troisième séquence
    // 1 passe au rouge
    digitalWrite(led_jaune_feux_1, LOW);
    digitalWrite(led_rouge_feux_1, HIGH);
    // 2 reste rouge
```

Exercice : Programmation de feux rouges
Première problématique

Test du programme

```
    delay(1000);
// quatrième séquence
    // 2 passe au vert
    digitalWrite(led_verte_feux_2, HIGH);
    digitalWrite(led_rouge_feux_2, LOW);
    // 1 reste au rouge
    delay(5000);
// cinquième séquence
    // 2 passe au orange
    digitalWrite(led_jaune_feux_2, HIGH);
    digitalWrite(led_verte_feux_2, LOW);
    // 1 reste rouge
    delay(2000);
// sixième séquence
    // 2 passe au rouge
    digitalWrite(led_jaune_feux_2, LOW);
    digitalWrite(led_rouge_feux_2, HIGH);
    // 1 reste rouge
}
```

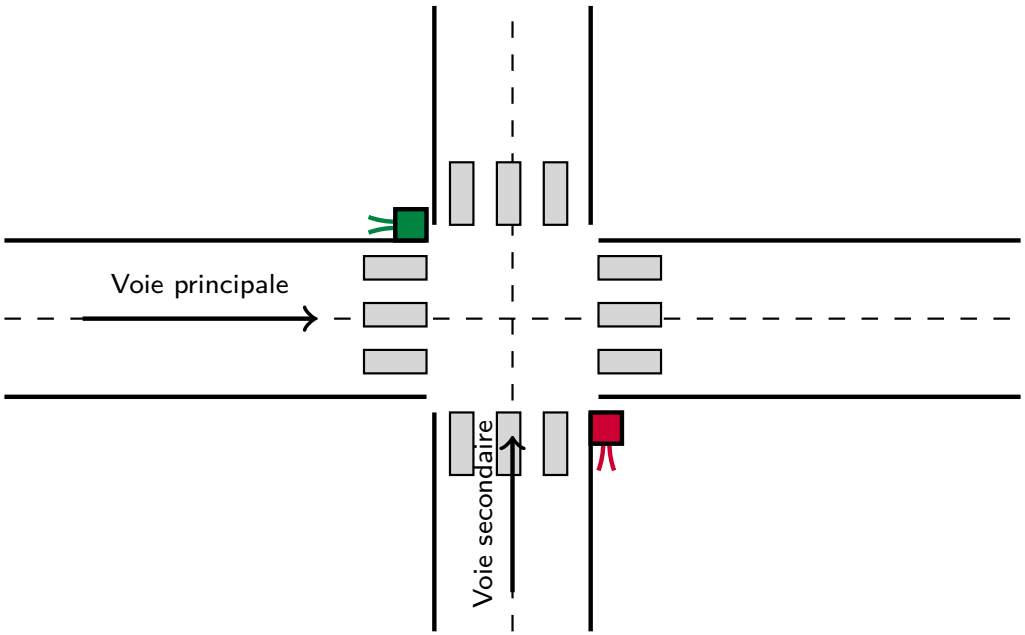
Exercice : Programmation de feux rouges

Seconde problématique

Objectif

Programmer un ensemble urbain avec une modélisation des feux de croisements sur carte Arduino.

On considère deux voies de circulation à sens unique avec un croisement comportant donc deux feux tricolores.



Exercice : Programmation de feux rouges

Seconde problématique

Problème 2 De nuit, la programmation souhaitée est la suivante :

- le feu principal reste au vert ;
- si une voiture est détectée au niveau du feu secondaire sans qu'aucune voiture ne soit présente au niveau du feu principal ;
  - le feu principal passe au orange durant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;
  - le feu secondaire passe au vert durant 5 secondes ;
  - il passe alors au orange pendant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;
- si une voiture est détectée au niveau du feu secondaire et qu'une voiture est également détectée au niveau du feu principal ;
  - le feu principal reste vert durant 5 secondes ;
  - le feu principal passe au orange durant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;
  - le feu secondaire passe au vert durant 5 secondes ;
  - il passe alors au orange pendant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;

Réaliser une programmation et un montage Arduino permettant de respecter cette coordination de feux de croisement.

Exercice : Programmation de feux rouges

Seconde problématique

Problème 2 De nuit, la programmation souhaitée est la suivante :

- le feu principal reste au vert ;
- si une voiture est détectée au niveau du feu secondaire sans qu'aucune voiture ne soit présente au niveau du feu principal ;
  - le feu principal passe au orange durant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;
  - le feu secondaire passe au vert durant 5 secondes ;
  - il passe alors au orange pendant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;
- si une voiture est détectée au niveau du feu secondaire et qu'une voiture est également détectée au niveau du feu principal ;
  - le feu principal reste vert durant 5 secondes ;
  - le feu principal passe au orange durant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;
  - le feu secondaire passe au vert durant 5 secondes ;
  - il passe alors au orange pendant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;

Réaliser une programmation et un montage Arduino permettant de respecter cette coordination de feux de croisement.

Exercice : Programmation de feux rouges
Seconde problématique

Problème 2 De nuit, la programmation souhaitée est la suivante :

- le feu principal reste au vert ;
- si une voiture est détectée au niveau du feu secondaire sans qu'aucune voiture ne soit présente au niveau du feu principal ;
  - le feu principal passe au orange durant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;
  - le feu secondaire passe au vert durant 5 secondes ;
  - il passe alors au orange pendant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;
- si une voiture est détectée au niveau du feu secondaire et qu'une voiture est également détectée au niveau du feu principal ;
  - le feu principal reste vert durant 5 secondes ;
  - le feu principal passe au orange durant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;
  - le feu secondaire passe au vert durant 5 secondes ;
  - il passe alors au orange pendant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;

Réaliser une programmation et un montage Arduino permettant de respecter cette coordination de feux de croisement.

Exercice : Programmation de feux rouges
Seconde problématique

Problème 2 De nuit, la programmation souhaitée est la suivante :

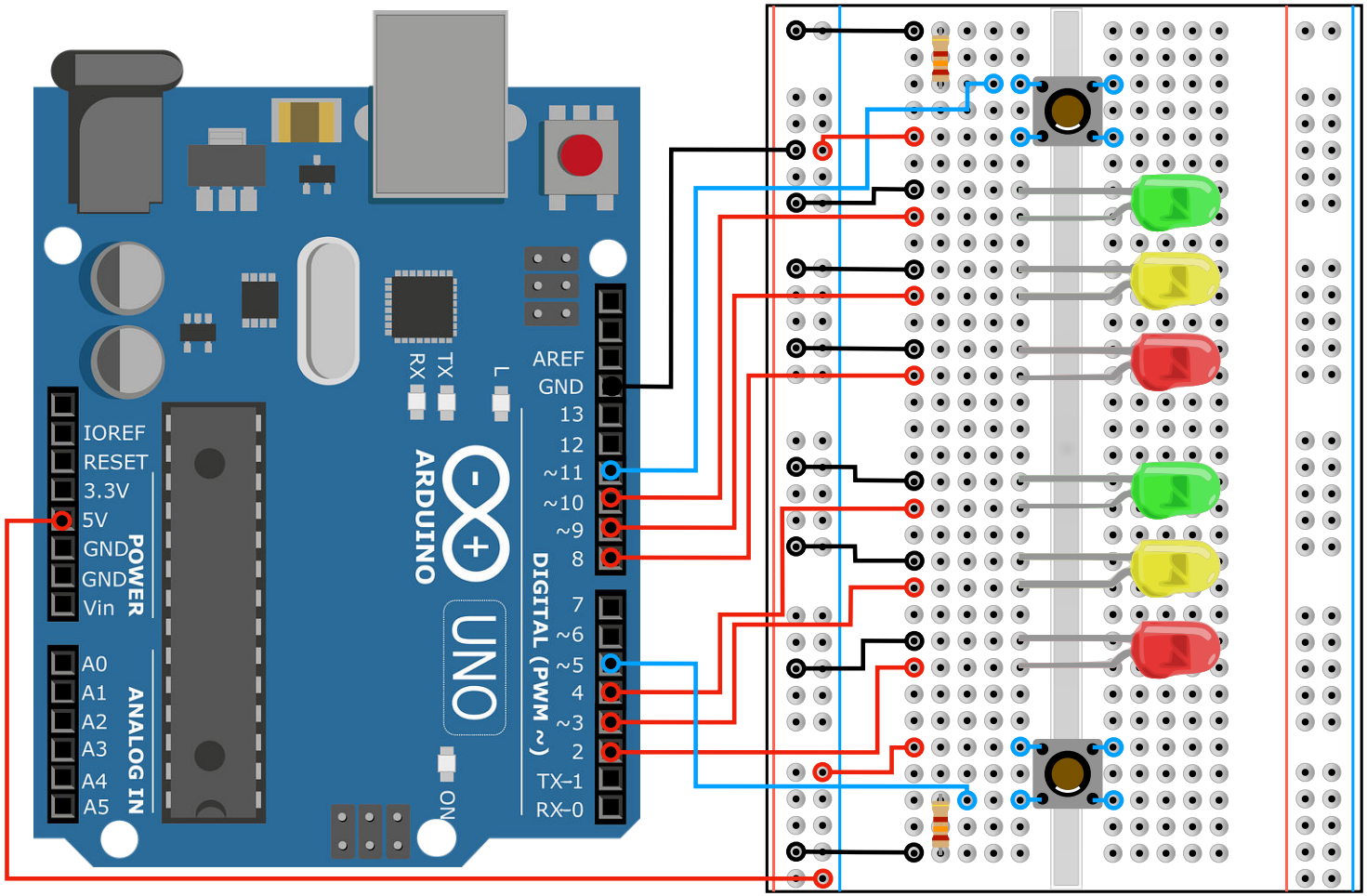
- le feu principal reste au vert ;
- si une voiture est détectée au niveau du feu secondaire sans qu'aucune voiture ne soit présente au niveau du feu principal ;
  - le feu principal passe au orange durant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;
  - le feu secondaire passe au vert durant 5 secondes ;
  - il passe alors au orange pendant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;
- si une voiture est détectée au niveau du feu secondaire et qu'une voiture est également détectée au niveau du feu principal ;
  - le feu principal reste vert durant 5 secondes ;
  - le feu principal passe au orange durant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;
  - le feu secondaire passe au vert durant 5 secondes ;
  - il passe alors au orange pendant 2 secondes ;
  - les deux feux restent au rouge durant 1 seconde ;

Réaliser une programmation et un montage Arduino permettant de respecter cette coordination de feux de croisement.

Exercice : Programmation de feux rouges

Seconde problématique

Branchement à réaliser



Exercice : Programmation de feux rouges

Seconde problématique

Test du programme

```
// définition des broches
const int led_rouge_feux_1 = 2;
const int led_jaune_feux_1 = 3;
const int led_verte_feux_1 = 4;
const int led_rouge_feux_2 = 8;
const int led_jaune_feux_2 = 9;
const int led_verte_feux_2 = 10;
int arriveeVoiture1=0;
int arriveeVoiture2=0;
```

Exercice : Programmation de feux rouges

Seconde problématique

Test du programme

```
void setup()
{
    // initialisation en sortie de toutes les broches
    pinMode(led_rouge_feux_1, OUTPUT);
    pinMode(led_jaune_feux_1, OUTPUT);
    pinMode(led_verte_feux_1, OUTPUT);
    pinMode(led_rouge_feux_2, OUTPUT);
    pinMode(led_jaune_feux_2, OUTPUT);
    pinMode(led_verte_feux_2, OUTPUT);
    // on initialise toutes les LED éteintes au début du programme
    // (sauf les deux feux rouges)
    digitalWrite(led_rouge_feux_1, LOW);
    digitalWrite(led_jaune_feux_1, LOW);
    digitalWrite(led_verte_feux_1, HIGH);
    digitalWrite(led_rouge_feux_2, HIGH);
    digitalWrite(led_jaune_feux_2, LOW);
    digitalWrite(led_verte_feux_2, LOW);
  // Déclaration des deux entrées utilisées
    pinMode(5,INPUT);
    pinMode(11,INPUT);
}
```

Exercice : Programmation de feux rouges

Seconde problématique

Test du programme

```
void loop()
{
    arriveeVoiture2 = digitalRead(5);
    arriveeVoiture1 = digitalRead(11);

    if (arriveeVoiture2 == HIGH && arriveeVoiture1 == LOW) {
      digitalWrite(led_verte_feux_1, LOW);
      digitalWrite(led_jaune_feux_1, HIGH);
      delay(2000);
      digitalWrite(led_rouge_feux_1, HIGH);
      digitalWrite(led_jaune_feux_1, LOW);
      delay(1000);
      digitalWrite(led_rouge_feux_2, LOW);
      digitalWrite(led_verte_feux_2, HIGH);
      delay(5000);
      digitalWrite(led_verte_feux_2, LOW);
      digitalWrite(led_jaune_feux_2, HIGH);
      delay(2000);
      digitalWrite(led_rouge_feux_2, HIGH);
      digitalWrite(led_jaune_feux_2, LOW);
      delay(1000);
      digitalWrite(led_rouge_feux_1, LOW);
      digitalWrite(led_verte_feux_1, HIGH);
    }
```

Exercice : Programmation de feux rouges

Seconde problématique

Test du programme

```
else if (arriveeVoiture2 == HIGH && arriveeVoiture1 == HIGH) {
  delay(10000);
  digitalWrite(led_verte_feux_1, LOW);
  digitalWrite(led_jaune_feux_1, HIGH);
  delay(2000);
  digitalWrite(led_rouge_feux_1, HIGH);
  digitalWrite(led_jaune_feux_1, LOW);
  delay(1000);
  digitalWrite(led_rouge_feux_2, LOW);
  digitalWrite(led_verte_feux_2, HIGH);
  delay(5000);
  digitalWrite(led_verte_feux_2, LOW);
  digitalWrite(led_jaune_feux_2, HIGH);
  delay(2000);
  digitalWrite(led_rouge_feux_2, HIGH);
  digitalWrite(led_jaune_feux_2, LOW);
  delay(1000);
  digitalWrite(led_rouge_feux_1, LOW);
  digitalWrite(led_verte_feux_1, HIGH);
}
```

Exercice : Programmation de feux rouges

Seconde problématique

Test du programme

```
    else if (arriveeVoiture2 == LOW && arriveeVoiture1 == HIGH){
      digitalWrite(led_rouge_feux_1, LOW);
      digitalWrite(led_verte_feux_1, HIGH);
      delay(1000);
    }
    else
    {
      digitalWrite(led_rouge_feux_1, LOW);
      digitalWrite(led_verte_feux_1, HIGH);
      delay(1000);
    }
}
```

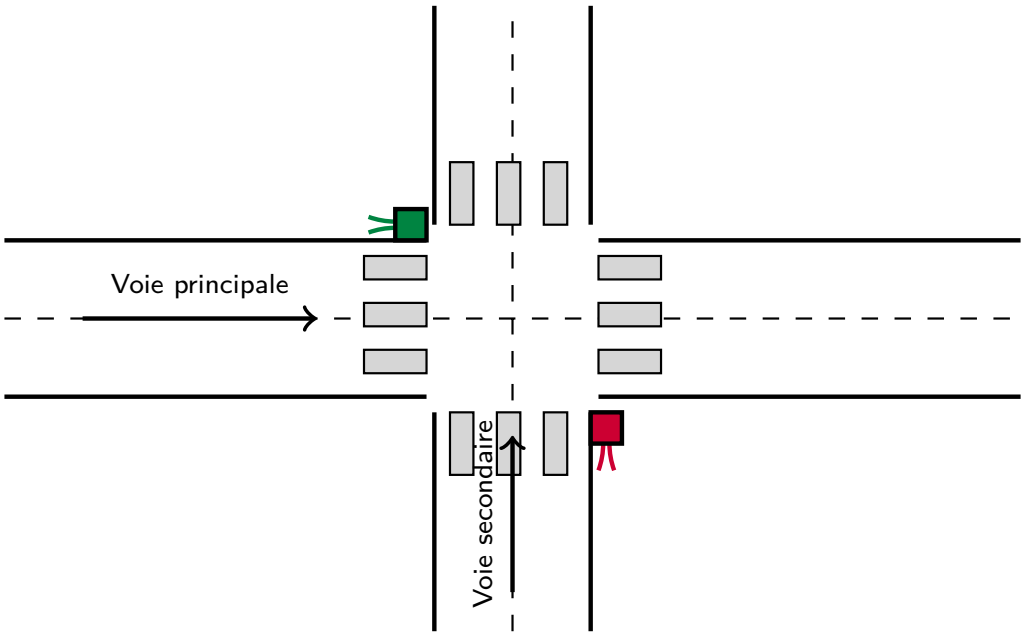
Exercice : Programmation de feux rouges

Troisième problématique

Objectif

Programmer un ensemble urbain avec une modélisation des feux de croisement sur carte Arduino.

On considère deux voies de circulation à sens unique avec un croisement comportant donc deux feux tricolores.



Exercice : Programmation de feux rouges

Troisième problématique

Problème 3 :
Il serait utile d'afficher en direct la présence ou non de véhicule afin de vérifier la cohérence de la programmation précédente. Pour cela, il sera considéré que deux capteurs sont utilisés et leur état est modifié en fonction de la boucle dans laquelle on se trouve. Le branchement reste le même, les modifications sont à réaliser au niveau de la programmation Arduino.

Exercice : Programmation de feux rouges

Troisième problématique

Test du programme

```
// définition des broches
const int led_rouge_feux_1 = 2;
const int led_jaune_feux_1 = 3;
const int led_verte_feux_1 = 4;
const int led_rouge_feux_2 = 8;
const int led_jaune_feux_2 = 9;
const int led_verte_feux_2 = 10;
int arriveeVoiture1=0;
int arriveeVoiture2=0;
// les index de tableau commencent à0.
const char * EtatCapteur1Text[] =
{"Absence route principal",
"Présence route principal"
};
const char * EtatCapteur2Text[] =
{"Absence route secondaire",
"Présence route secondaire"
};
int EtatCapteur1 = 0;
int EtatCapteur2 = 0;
```

Exercice : Programmation de feux rouges

Troisième problématique

- Test du programme : ajout d'un `Serial.begin` pour l'affichage de la valeur souhaitée

```
void setup()
{
    // ouvre le port série et fixe le debit
    // de communication a 115200 bauds
    Serial.begin(115200);
    // initialisation en sortie de toutes les broches
    pinMode(led_rouge_feux_1, OUTPUT);
    pinMode(led_jaune_feux_1, OUTPUT);
    pinMode(led_verte_feux_1, OUTPUT);
    pinMode(led_rouge_feux_2, OUTPUT);
  ...
```

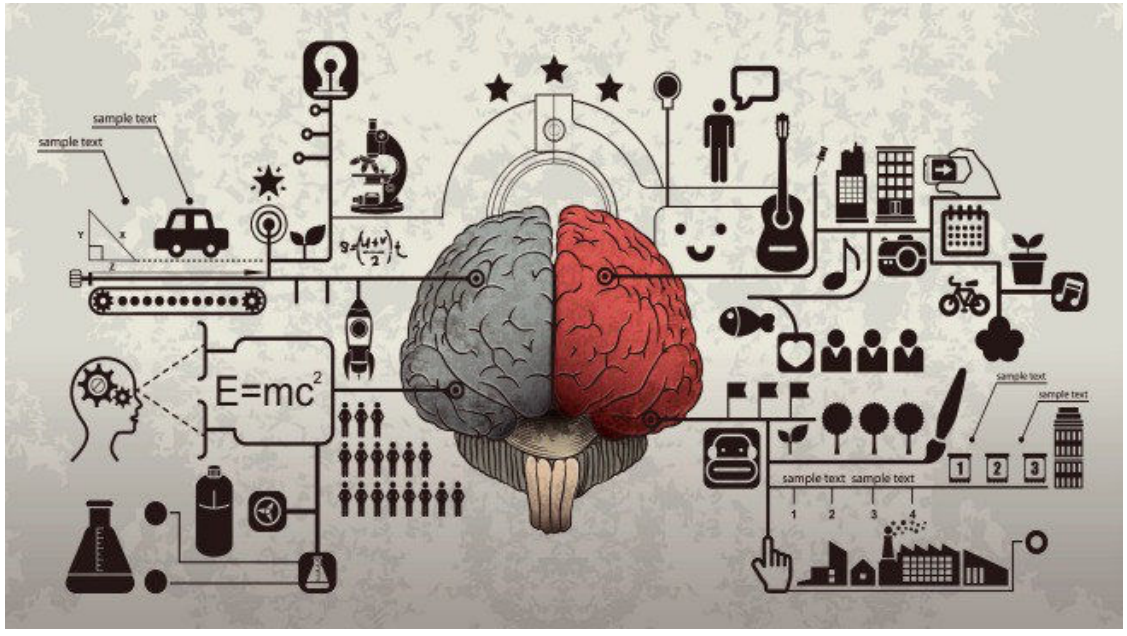
Exercice : Programmation de feux rouges

Troisième problématique

- Test du programme : association d'une valeur à l'état du capteur et envoie de la réponse au moniteur série

```
void loop()
{
    arriveeVoiture2 = digitalRead(5);
    arriveeVoiture1 = digitalRead(6);

    if (arriveeVoiture2 == HIGH && arriveeVoiture1 == LOW) {
      // Affectation d'une valeur aux états du capteur
    EtatCapteur1=0;
      EtatCapteur2=1;
      // Tracé dans le moniteur de l'état obtenu
      Serial.println(EtatCapteur1Text[EtatCapteur1]);
      Serial.println(EtatCapteur2Text[EtatCapteur2]);
      digitalWrite(led_verte_feux_1, LOW);
    ...
```



Virgile Lacharnay <virgile.lacharnay@gmail.com>
Ce document évolue. Merci de signaler toute erreur ou coquille.