

## Cours : Découverte de l'Arduino.



Ressources :

Stephane coiffier / La carte Arduino Uno.

Lycée polyvalent Bellevue de Toulouse



# COURS STI2D

## La programmation des cartes Arduino Uno.

2

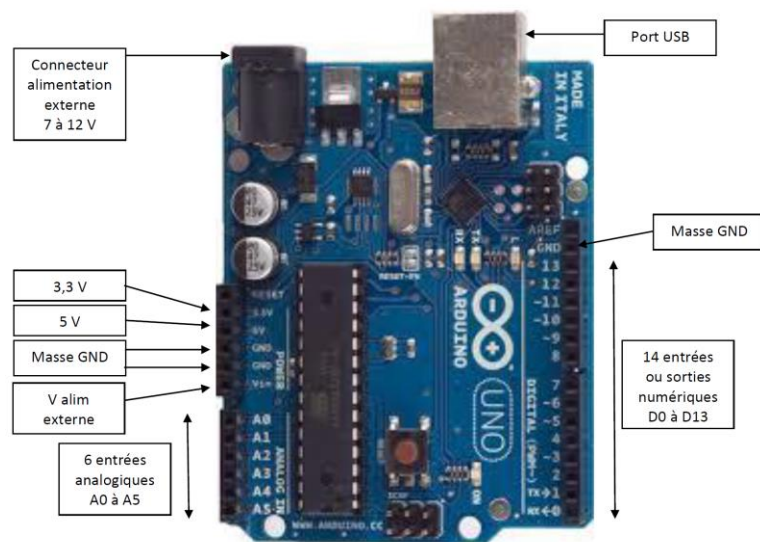
### Table des matières

1	Présentation.....	3
1.1	Les entrées et sorties numériques.....	3
1.2	Les entrées analogiques.....	3
1.3	Le microcontrôleur.....	4
1.4	Les broches d'alimentation.....	4
1.5	Le connecteur USB.....	4
1.6	Le jack d'alimentation.....	5
1.7	Les LED.....	5
1.8	Le bouton RESET.....	5
2	Les programmes.....	5
3	Le logiciel que nous allons utiliser.....	6
3.1	Configuration du logiciel.....	7
4	Début de la programmation.....	7
4.1	La structure d'un programme.....	8
4.2	La syntaxe du langage ARDUINO.....	8
4.3	Les commentaires.....	8
4.4	Les données variables et constantes.....	9
4.5	Les opérateurs.....	9
4.6	Les bibliothèques de fonctions.....	10
4.7	Les fonctions de gestion du temps.....	11
4.8	Les fonctions de gestion des E/S numériques.....	12
4.9	Les fonctions de gestion des sorties PWM.....	12
4.10	Les fonctions de gestion des entrées analogiques.....	13
4.11	Les fonctions particulières de gestion des entrées - sorties.....	13
4.12	Les fonctions de manipulation de bits.....	14
4.13	Les fonctions de gestions du port série asynchrone.....	15

## 1 Présentation.

Il existe différentes cartes Arduino, la Uno, la Nano, La Méga... nous allons nous attarder dans ce cours à la carte Arduino Uno R3 (R3 car c'est la 3<sup>ème</sup> évolution ou Révision de la carte pour faire évoluer ses performances).

La carte Arduino Uno



### 1.1 Les entrées et sorties numériques.

Cette carte dispose de 14 entrées ou sorties numériques : D0 à D13.

Les signaux véhiculés par ces connecteurs ne peuvent prendre que deux états HAUT (5 Volts) ou BAS (0 Volt) courant de 40 mA maximum par sortie.

Les connecteurs D0 et D1 sont réservés pour la liaison USB et ne sont donc pas utilisés (RX et TX sont utilisés pour gérer les flux de données entrants et sortants)

Les connecteurs D3, D5, D6, D9, D10, et D11 repérés par un  $\sim$ , peuvent être utilisés en sortie PWM (Pulse With Modulation ou en français Modulation de Largeur d'Impulsion), pour faire varier la luminosité d'une DEL ou la vitesse d'un moteur.

Les sorties PWM peuvent avoir 2<sup>8</sup> valeurs, soit 256 allant de 0 à 255.

### 1.2 Les entrées analogiques.

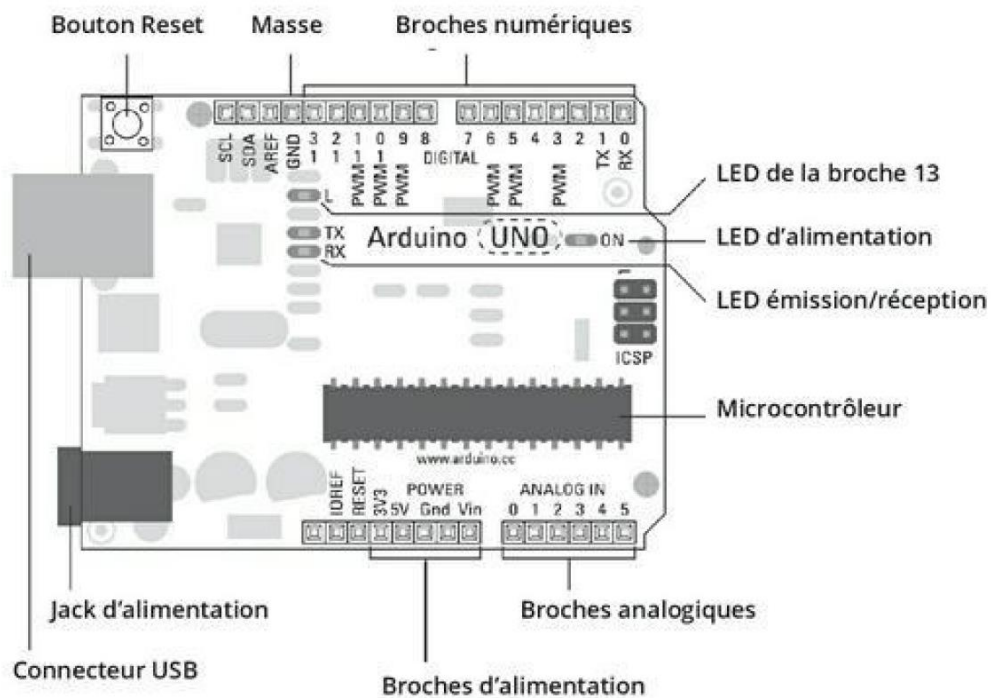
6 entrées analogiques : A0, A1, A2, A3, A4 et A5.

Ces entrées analogiques peuvent servir à recevoir les informations de capteurs comme les photorésistance, des potentiomètres, des capteurs de températures...

Les entrées analogiques peuvent avoir 2<sup>10</sup> valeurs, soit 1024, allant de 0 à 1023.

### 1.3 Le microcontrôleur.

Le microcontrôleur est en quelque sorte le cerveau de la carte. C'est un ATmega328, fabriqué par la société Atmel. C'est le gros circuit intégré noir qui se trouve sur la carte.



### 1.4 Les broches d'alimentation.

Les broches d'alimentation servent à distribuer de l'énergie aux circuits et aux entrées sorties.

La broche Vin (Voltage In ou tension d'entrée en français) peut être utilisée comme source de courant équivalente à celle utilisée par le connecteur d'alimentation. Vous pouvez aussi utiliser cette broche pour alimenter l'Arduino depuis une autre source.

Les trois broches GND (Ground, Masse) sont celles de masse ou de terre, essentielles pour fermer le circuit de l'énergie.

Il y a une broche de 5 V pour alimenter les composants ou les circuits en 5 volts.

Une broche 3,3 V pour alimenter les composants et les circuits qui ne peuvent pas fonctionner en 5 V.

### 1.5 Le connecteur USB.

Pour indiquer au microcontrôleur ce qu'il doit faire, il faut pouvoir lui envoyer un programme. C'est par l'intermédiaire du connecteur USB que nous allons relier la carte Arduino Uno à l'ordinateur. Ce connecteur sert également à alimenter en énergie la carte.



# COURS STI2D

## La programmation des cartes Arduino Uno.

5

### 1.6 Le jack d'alimentation.

A côté du connecteur USB, se trouve un connecteur qui sert à une alimentation externe pour répondre à des besoins de puissance qui dépassent les possibilités du port USB. On peut y connecter un adaptateur AC-DC, une batterie ou même un panneau solaire.

Le connecteur est du type jack de 2,1mm avec positif au centre. Le Jack a une partie externe et une partie interne, la partie interne doit être reliée au pôle positif de l'alimentation.

**Si vous connectez une alimentation dans le sens inverse (négatif au centre), vous produisez une « inversion de polarité » et il y a de grandes chances que la carte soit détruite.**

La tension recommandée par la carte Uno R3 est comprise entre 7 V et 12 V. Si la carte ne reçoit pas assez de courant, elle risque de ne pas fonctionner correctement.

### 1.7 Les LED.

La carte dispose de quatre LED nommées L, RX, TX, et ON.

- ON est verte et signifie que la carte est alimentée.
- RX et TX ne s'allument que lorsque des données sont reçues ou émises par la carte.
- L est une LED particulière elle est directement connectée à la broche 13 de la carte et permet de procéder à des tests.

### 1.8 Le bouton RESET.

Une carte Arduino dispose d'un bouton à côté du connecteur USB. C'est le bouton de réinitialisation (RESET). Il réinitialise l'Arduino ou l'arrête complètement lorsqu'il est maintenu appuyé un certain temps.

## 2 Les programmes.

Une carte Arduino Uno est une carte électronique programmable, elle ne sait rien faire, elle a besoin d'un programme pour fonctionner.

Un programme est une suite d'instruction qui est exécutée par un système.

Exemple de programme :

Un programme est une suite de mots, de lignes de code qui seront ensuite traduits en langage machine via un compilateur afin que le microcontrôleur puisse les comprendre.

Le langage machine c'est du binaire, un microprocesseur ne comprend des 0 et des 1...

```
/* Clignotement de 2 Leds alternées
 * séquence 4 STI2D / exercice 2
 * DEVOS
 */
int LED1 = 12;
int LED2 = 11;

void setup() {
  // put your setup code here, to run once:
  pinMode(LED1,OUTPUT);
  pinMode(LED2,OUTPUT);
}

void loop() {
  // put your main code here, to run repeated:
  digitalWrite(LED1,HIGH);
  digitalWrite(LED2,LOW);
  delay(500);

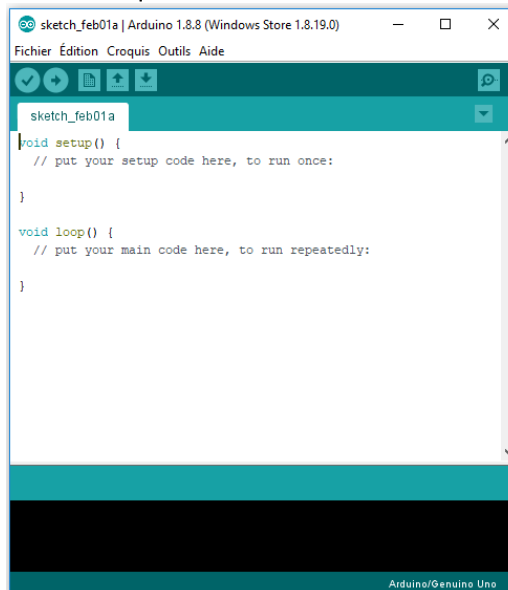
  digitalWrite(LED2,HIGH);
  digitalWrite(LED1,LOW);
  delay(500);
}
```

que

### 3 Le logiciel que nous allons utiliser.

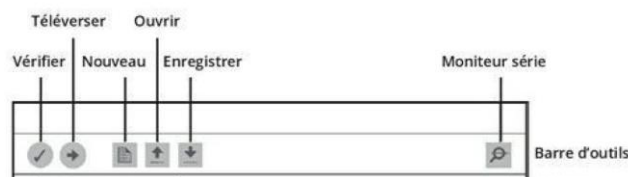
Pour écrire les programmes, nous allons utiliser le logiciel Arduino. C'est un environnement de développement intégré (un EDI) qui se présente sous la forme d'une interface graphique.

Un programme en Arduino s'appelle un « Croquis ».



Cet EDI est divisé en 4 zones principales :

- **La barre de Menus** : comme dans la plupart des logiciels cette barres contient les menus « Fichier », « Edition », « Croquis », « Outils » et « Aide ».
- **La barre d'outils** contient plusieurs boutons qui sont utilisés lorsque l'on écrit des programmes pour Arduino.



- ❖ **Vérifier** : permet de compiler votre croquis. Si votre code contient des erreurs, le compilateur vous indiquera où elles se trouvent.
- ❖ **Téléverser** : Envoie votre croquis à la carte Arduino connectée à l'ordinateur. Si votre croquis n'a pas été compilé auparavant, il le sera avant d'être envoyé.
- ❖ **Nouveau** : crée un nouveau croquis
- ❖ **Ouvrir** : Ouvre un croquis existant.
- ❖ **Enregistrer** : Sauvegarde le croquis courant.
- ❖ **Moniteur série** : Vous permet de visualiser les données qui sont envoyées ou reçues par votre carte Arduino.
- **Editeur de texte** : Cette zone affiche votre croquis sous la forme de texte.
- **Zone des messages** : C'est dans cette zone que seront indiqués les erreurs lorsque que votre compilateur détectera des erreurs.

### 3.1 Configuration du logiciel.

Avant de commencer à faire le premier programme, il faut que notre logiciel reconnaisse notre carte pour pouvoir communiquer avec elle. Pour cela vous devez lui indiquer quel type de carte vous avez en ouvrant le menu « Outils » et « Type de carte » sélectionnez : Arduino / Genuino Uno.

Puis il faut vérifier que le Port de communication avec la carte est bien sélectionné. Allez sur le menu « Outils » et sélectionnez le bon « Port Com ».

## 4 Début de la programmation.

Nous allons commencer un programme simple pour faire clignoter une LED. Pour cela il va vous falloir :

- Une breadboard
- Une LED
- Une résistance de 220  $\Omega$  minimum. (Regarder la ressource sur le calcul d'une valeur de résistance).
- La carte Arduino Uno
- Des fils

```
01_cligno_led
/* Croquis pour faire clignoter une led connectée sur la broche 12 d'une arduino Uno.
 * La broche 12 sera connectée à une led q1 sera branchée en série avec une résistance de 330 Ohms
 * |
 */

/*Définition de la variable Led qui contient le numéro de la broche
 * de la carte où va être branché la Led.
 */
int led = 12;

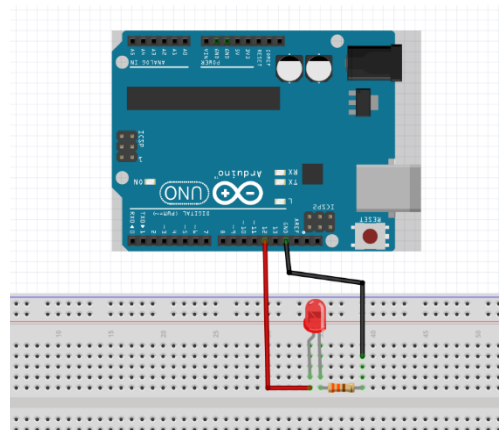
void setup() {
  // On configure la broche 12 en sortie
  pinMode(led, OUTPUT);
}

void loop() {
  // Ecriture du niveau logique 1 sur la broche 12 afin d'allumer la led
  digitalWrite(led, HIGH);

  // attendre 1 seconde
  delay(1000);

  //Ecriture du niveau logique 0 sur la broche 12 afin d'éteindre la led
  digitalWrite(led, LOW);

  // attendre 1 seconde
  delay(1000);
}
```



Le premier programme est terminé, il faut maintenant le compiler.

### 4.1 La structure d'un programme

Un programme destiné à une carte ARDUINO est constitué de **3 parties** :

Zone de définition des constantes ou des variables ou d'inclusion des bibliothèques

/\* Ce programme fait clignoter une LED branchée sur la broche n°13 et fait également clignoter la diode de test de la carte \*/

```
int ledPin = 13;           // LED connectée à la broche n°13
```

```
void setup()
{
    pinMode(ledPin,OUTPUT) ;    // Définit la broche n°13 comme une sortie
}
```

```
void loop()
{
    digitalWrite(ledPin,HIGH) ;    // Met la sortie ledPin au NL1 (diode allumée)
    delay(3000) ;                  // Attendre 3 s
    digitalWrite(ledPin,LOW) ;     // Met la sortie ledPin au NL0 (diode éteinte)
    delay(1000) ;                  // Attendre 1 s
}
```

Fonction setup()  
qui contient les instructions d'initialisation

Fonction loop()  
qui contient les instructions du programme

La **première partie** permet de **définir les constantes et les variables** en déclarant leur type. Elle permet également l'inclusion des bibliothèques utilisées dans le programme au moyen de #include.

La **fonction setup()** contient les **instructions d'initialisation ou de configuration des ressources** de la carte comme par exemple, la configuration en entrée ou sorties des broches d'E/S numériques, la définition de la vitesse de communication de l'interface série, etc.. Cette fonction **n'est exécutée qu'une seule fois** juste après le lancement du programme.

La **fonction loop()** contient les **instructions du programme** à proprement parlé. Cette fonction sera **répétée indéfiniment** tant que la carte ARDUINO restera sous tension.

### 4.2 La syntaxe du langage ARDUINO

La cinquantaine d'éléments de la syntaxe ARDUINO est visible ici <http://www.arduino.cc/en/Reference/HomePage> ainsi qu'à partir du document "index.html" (dans le dossier "Reference" que vous avez téléchargé avec ARDUINO), également accessible dans le menu "Aide" du logiciel.

### 4.3 Les commentaires.

Pour placer des commentaires sur une ligne unique ou en fin de ligne, il faut utiliser la syntaxe suivante :

```
// Cette ligne est un commentaire sur UNE SEULE ligne
```



Pour placer des commentaires sur plusieurs lignes :

```
/* Commentaire, sur PLUSIEURS lignes qui sera ignoré par le programme, mais
pas par celui qui lit le code */
```

### 4.4 Les données variables et constantes.

Les différents types utilisés avec la programmation ARDUINO sont :

Nom	Description
<b>boolean</b>	Donnée logique (true ou false) sur 8 bits
<b>byte</b>	Entier non signé sur 8 bits
<b>int</b>	Entier signé sur 16 bits
<b>unsigned int</b>	Entier non signé sur 16 bits
<b>long</b>	Entier signé sur 32 bits
<b>unsigned long</b>	Entier non signé sur 32 bits
<b>float</b>	Nombre à virgule flottant sur 32 bits
<b>char</b>	Caractère sur 8 bits. Seuls les caractères ayant pour code ASCII une valeur 0 à 127 sont définis

Une constante peut être définie au moyen de **const** ou de **#define** :

```
#define N 2 //Toute variable N rencontrée dans le programme sera remplacée par la valeur 2
const byte N=2 // N est constante de type « byte » et de valeur 2
```

Un certain nombre de noms de constantes sont prédéfinis dans le langage Arduino.

Nom	Description
<b>true</b>	Donnée binaire égale à 1 ou donnée numérique différente de 0
<b>false</b>	Donnée binaire ou numérique égale à 0
<b>HIGH</b>	Génération d'un niveau de tension haut sur une des sorties numériques
<b>LOW</b>	Génération d'un niveau de tension bas sur une des sorties numériques
<b>INPUT</b>	Configuration d'une broche numérique en entrée
<b>OUTPUT</b>	Configuration d'une broche numérique en sortie

### 4.5 Les opérateurs.

Les opérateurs arithmétiques :

Symbole	Description	Exemple
<b>+</b>	Addition	c = a + b ;
<b>-</b>	Soustraction	c = a - b ;
<b>*</b>	Multiplication	c = a * b ;
<b>/</b>	Division	c = a / b ;
<b>%</b>	Modulo (reste de la division entière)	c = a % b ;

### Les opérateurs logiques :

Symbole	Description	Exemple
&	ET	c = a & b ;
	OU	c = a   b ;
^	OU exclusif	c = a ^ b ;
~	NON	b = ~ a ;
>>	Décalage à droite des bits	c = a >> b ; //a décalé b fois à droite
<<	Décalage à gauche des bits	c = a << b ; //a décalé b fois à gauche

### Les affectations :

Symbole	Description	Exemple
=	Affectation ordinaire	a = b ;
+=	Additionner de	a += b ; → a = a + b ;
-=	Soustraire de	a -= b ; → a = a - b ;
*=	Multiplier par	a *= b ; → a = a * b ;
/=	Diviser par	a /= b ; → a = a / b ;
%=	Reste de la division entière par	a %= b ; → a = a % b ;
&=	ET avec	a &= b ; → a = a & b ;
=	OU avec	a  = b ; → a = a   b ;
--	Soustraire de 1 (décrément)	a -- ; → a = a - 1 ;
++	Additionner de 1 (incrément)	a ++ ; → a = a + 1 ;

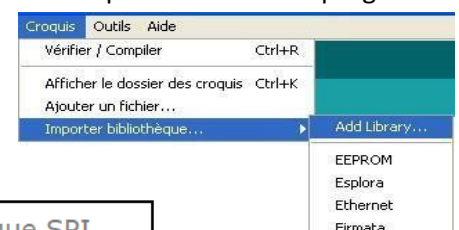
### Les opérateurs de comparaisons :

Symbole	Description	Exemple
&&	ET logique	if (a && b)
	OU logique	if (a    b)
==	Égal à	if (a == b)
!=	Différent de	if (a != b)
>	Supérieur à	if (a > b)
<	Inférieur à	if (a < b)
>=	Supérieur ou égal à	if (a >= b)
<=	Inférieur ou égal à	if (a <= b)

## 4.6 Les bibliothèques de fonctions

Une bibliothèque est un ensemble de fonctions utilitaires mises à disposition des utilisateurs de l'environnement Arduino. Les fonctions sont regroupées en fonction de leur appartenance à un même domaine conceptuel (mathématique, graphique, tris, etc).

L'IDE ARDUINO comporte par défaut plusieurs bibliothèques externes. Pour les importer dans votre programme, vous devez utiliser le menu suivant :



L'instruction suivante sera alors ajoutée au début de votre programme :

#include <la\_bibliothèque.h>

Exemple :

```
#include <SPI.h> // Importation de la bibliothèque SPI
```

### 4.7 Les fonctions de gestion du temps.

#### La fonction « delay »

`delay(tempo);`

Cette fonction **génère une pause** dont la durée est égale à  $\text{tempo} \times 1 \text{ ms}$ .

La variable tempo doit être de type unsigned long ce qui peut autoriser des temporisations qui peuvent être très longues.

`delay(1000);`     // Pause de 1 s

#### La fonction « delayMicroseconds »

`delayMicroseconds(tempo);`

Cette fonction **génère une pause** dont la durée est égale à  $\text{tempo} \times 1 \mu\text{s}$ .

La variable tempo doit être de type unsigned int.



La durée de la pause n'est assurée que des valeurs supérieures à  $3 \mu\text{s}$ .

`delayMicroseconds(100);`     // Pause de  $100 \mu\text{s}$

#### La fonction « millis() »

Valeur = `millis()` ;

Cette fonction **retourne le nombre de millisecondes depuis le démarrage du programme** par la carte ARDUINO.

La valeur retournée est de type unsigned long. Le nombre de millisecondes retourne à zéro environ après 50 jours.

#### La fonction « micros() »

Valeur = `micros()` ;

Cette fonction **retourne le nombre de microsecondes depuis le démarrage du programme** par la carte ARDUINO.

La valeur retournée est de type unsigned long



Cette fonction a une résolution de  $4 \mu\text{s}$  (avec une horloge de 16MHz), La valeur retournée est donc un multiple de 4.

Le nombre de millisecondes retourne à zéro environ après 70 minutes.

### 4.8 Les fonctions de gestion des E/S numériques.

#### La fonction « pinMode »

`pinMode`(Numéro broche, Sens de fonctionnement) ;

Cette fonction permet de **configurer le sens de fonctionnement**, entrée ou sortie, **de la broche sélectionnée**.

`pinMode` (12, `OUTPUT`) ; // La broche 12 est configurée comme une sortie

#### La fonction « digitalWrite »

`digitalWrite`(Numéro broche, Niveau logique) ;

Cette fonction permet **d'imposer un niveau logique** haut ou bas **sur la sortie numérique sélectionnée**.

`digitalWrite`(12, `HIGH`) ; // La sortie 12 est placée au niveau logique haut (NL1)

#### La fonction « digitalRead »

Valeur = `digitalRead`(Numéro broche) ;

Cette fonction permet de **lire le niveau logique sur l'entrée numérique sélectionnée**. Cette fonction ne retourne que deux valeur HIGH ou LOW.

etat = `digitalRead`(11) ; // La variable etat est égale au niveau logique de l'entrée 11

### 4.9 Les fonctions de gestion des sorties PWM.

Les  $\mu C$  qui équipent les cartes Arduino sont capables de générer des signaux à Modulation à Largeur d'Impulsions (MLI) ou « Pulse Width Modulation » (PWM).

Il s'agit de signaux logiques rectangulaires de fréquence égale à 490 Hz et à rapport cyclique programmable.

#### La fonction « analogWrite »

`analogWrite`(Numéro sortie, Rapport cyclique) ;

Cette fonction permet de **générer, sur la sortie sélectionnée, un signal PWM avec le rapport cyclique désiré**. La variable « Rapport cyclique » doit être de type int, c'est-à-dire qu'elle peut être comprise entre 0 et 255 pour un rapport cyclique, pour le signal généré, compris entre 0 et 100 %.

Le signal PWM est généré à partir de l'instant de l'exécution de la fonction « `analogWrite` » et jusqu'à ce qu'une nouvelle instruction « `analogWrite` » ou une instruction « `digitalRead` » ou « `digitalWrite` » soit exécutée sur la même broche.



La valeur du rapport cyclique n'est garantie que pour des valeurs supérieures à 10 %.

`analogWrite`(2,64) ; // Signal PWM de rapport cyclique de 25 % sur la sortie analogique 2

### 4.10 Les fonctions de gestion des entrées analogiques.

Les entrées analogiques sont connectées à un convertisseur analogique numérique (CAN) 10 bits. La sortie du CAN génère une donnée égale à 0 lorsque la tension d'entrée est nulle et une donnée égale à 1023 lorsque la tension d'entrée est égale à la tension de référence du CAN.

#### La fonction « analogReference »

`analogReference(Type);`

Cette fonction permet de **définir la tension de référence du CAN**.

La variable type peut prendre les valeurs suivantes :

- **DEFAULT** : La tension de référence est égale à la tension d'alimentation de la carte Arduino (5V pour la carte Arduino Uno) ;
- **INTERNAL** : La tension de référence est égale à 1,1 V pour la carte Arduino Uno ;
- **EXTERNAL** : La tension de référence est égale à la tension appliquée sur l'entrée externe AREF de la carte Arduino.



La tension appliquée sur une entrée analogique ne doit jamais être supérieure à la tension d'alimentation de la carte.

#### La fonction « analogRead »

Valeur = `analogRead(Numéro entrée);`

Cette fonction permet de **lire le résultat de la conversion analogique numérique de la tension présente sur l'entrée analogique sélectionnée**.

La donnée retournée étant comprise entre 0 et 1023, doit être de type int.



La durée de la conversion est d'environ 100 µs.

`tension = analogRead(1);` // tension est égale au résultat de la CAN de l'entrée analogique 1

### 4.11 Les fonctions particulières de gestion des entrées - sorties

#### La fonction « tone »

`tone(Numéro sortie, Fréquence, Durée);`

Cette fonction permet de **générer, sur la sortie sélectionnée, un signal carré** (rapport cyclique de 50 %) de fréquence programmable et pendant la durée désirée.

La donnée « Fréquence » doit être donnée en Hz et la donnée « Durée » doit être fixée en ms. Si cette durée n'est pas précisée, le signal est généré jusqu'à ce que la fonction « noTone » soit exécutée.

### La fonction « notone »

`notone`(Numéro sortie) ;

Cette fonction permet de **stopper la génération du signal carré sur la sortie sélectionnée.**


### La fonction « pulseIn »

durée = `pulseIn`(Numéro entrée, Type, Delai max) ;

Cette fonction permet de **mesurer la durée d'une impulsion sur l'entrée sélectionnée.** Le résultat retourné est du type unsigned long.

La valeur de la donnée « Type » peut être HIGH dans le cas d'une impulsion au NL1 ou LOW d'une impulsion au NLO.

La variable « Delai max » permet de définir le délai d'attente de l'impulsion. Lorsque ce délai est dépassé et qu'aucune impulsion ne s'est pas produite, la fonction retourne une valeur nulle. Cette donnée est du type unsigned long et est exprimé en  $\mu$ s. Si cette donnée n'est pas précisée, la valeur par défaut sera de 1 s.

 Le résultat fourni n'est considéré comme correct que pour des impulsions de durée comprise entre 10  $\mu$ s et 3 min.

## 4.12 Les fonctions de manipulation de bits.

### La fonction « lowByte »

resultat = `lowByte`(Donnée) ;

Cette fonction permet **d'extraire les 8 bits de poids faible de la variable « Donnée ».**

La valeur retournée est de type byte alors que la variable « Donnée » peut être de n'importe quel type (donc n'importe quelle taille).

### La fonction « highByte »

resultat = `highByte`(Donnée) ;

Cette fonction permet **d'extraire le deuxième octet en partant de la droite de la variable « Donnée ».**

La valeur retournée est de type byte alors que la variable « Donnée » peut être de n'importe quel type.

### La fonction « bitRead »

resultat = `bitRead`(Donnée, n) ;

Cette fonction permet de **lire le n<sup>ième</sup> bit de la variable « Donnée ».**

### La fonction « bitWrite »

`bitWrite`(Donnée, n, Niveau) ;

Cette fonction permet **d'imposer un niveau logique sur le nième bit de la variable « Donnée ».**



# COURS STI2D

## La programmation des cartes Arduino Uno.

15

### La fonction « bitSet »

`bitSet(Donnée, n) ;`

Cette fonction permet d'imposer un 1 logique sur le n<sup>ième</sup> bit de la variable « Donnée ».

### La fonction « bitClear »

`bitClear(Donnée, n) ;`

Cette fonction permet d'imposer un 0 logique sur le n<sup>ième</sup> bit de la variable « Donnée ».

## 4.13 Les fonctions de gestions du port série asynchrone.

La carte ARDUINO UNO dispose d'un port série asynchrone accessible via les E/S numériques 0 (ligne Rx) et 1 (Ligne Tx).

Le port série asynchrone des microcontrôleurs qui équipent les cartes ARDUINO disposent d'une mémoire tampon capable de mémoriser jusqu'à 128 caractères reçus.

### La fonction « Serial.begin »

`Serial.begin(Vitesse) ;`

Cette fonction qui doit être appelée au moins une fois, généralement dans la fonction `setup()`, permet de **définir la vitesse utilisée par la liaison série**.

La valeur prise par la variable « Vitesse » doit être une des vitesses définies par la norme RS232.

### La fonction « Serial.end »

`Serial.end() ;`

Cette fonction permet de **désactiver la liaison série asynchrone** et donc de libérer les broches numériques 0 et 1.

### La fonction « Serial.available »

`Serial.available() ;`

Cette fonction permet de **connaître le nombre de caractères reçus** et donc contenus dans la mémoire tampon en attente de lecture par le programme. La variable « nombre » est de type `int`.

### La fonction « Serial.read »

`Serial.available() ;`

Cette fonction permet de **lire le premier caractère disponible** dans la mémoire tampon de réception. La variable « caractère » est de type `int`. La valeur retournée est -1 si aucun caractère n'a été reçu (mémoire tampon vide).





# COURS STI2D

## La programmation des cartes Arduino Uno.

16

### La fonction « Serial.flush »

`Serial.flush()` ;

Cette fonction permet de **vider la mémoire tampon**.

### La fonction « Serial.write »

`Serial.write(Donnée)` ;

Cette fonction permet **d'émettre une ou plusieurs données sur la liaison série** sous la forme binaire brute. Si la donnée est de type chaîne de caractères, les caractères sont transmis, sous le format ASCII, les uns après les autres.

### La fonction « Serial.print »

`Serial.print(Donnée, format)` ;

Cette fonction permet **d'émettre une ou plusieurs données sur la liaison série en précisant le codage utilisé**.

Si la variable « Format » n'est pas précisée, une donnée numérique sera considérée comme décimale, une chaîne de caractères sera transmise tel quelle et une donnée à virgule flottante sera transmise uniquement avec deux décimales.

Sinon la variable « Format » permet de préciser le codage ou la base de numération pour les données numériques. Elle peut prendre une des valeurs suivantes : BIN (binaire), OCT (Octal), HEX (hexadécimal), DEC (Décimal) ou BYTE (codage ASCII). Pour les données à virgule flottante, la variable « Format » peut prendre une valeur numérique comprise entre 0 et 7 correspondant au nombre de décimales à transmettre.

`Serial.print(100101,BIN)` ; // Transmission de la donnée (100101)<sub>2</sub>

`Serial.print(4B,HEX)` ; // Transmission de la donnée (4B)<sub>16</sub>

`Serial.print(K,BYTE)` ; // Transmission de la donnée (K)<sub>ASCII</sub>

`Serial.print(1.23456,0)` ; // Transmission de la donnée 1

`Serial.print(1.23456,5)` ; // Transmission de la donnée 1.23456