

# Apprendre à recevoir des fonctions avec paramètres sur le port Série avec Arduino



## Ateliers Arduino

par X. HINAULT

[www.mon-club-elec.fr](http://www.mon-club-elec.fr)



Tous droits réservés – 2012.

**Ce document légèrement payant est soumis au droit d'auteur et est réservé à l'usage personnel.**

Afin d'encourager la production de supports didactiques de qualité, ce document est légèrement payant.

La licence d'utilisation est attribuée pour un usage personnel uniquement, dans le cercle familial. Mise en ligne et diffusion non autorisées.

Si vous n'êtes pas le détenteur de la licence attribuée pour l'usage de ce document, soyez sympa, merci d'acheter votre exemplaire personnel ici : <https://monclubelec.dpdcart.com/>

Pour tout problème lié à l'utilisation de ce document, veuillez envoyer une copie ici : [support@mon-club-elec.fr](mailto:support@mon-club-elec.fr)

Pour obtenir tout autres types de licence d'utilisation (enseignement, commercial, etc...), veuillez contacter l'auteur ici : [support@mon-club-elec.fr](mailto:support@mon-club-elec.fr)

Vous avez constaté une erreur ? une coquille ? N'hésitez pas à nous le signaler à cette adresse : [support@mon-club-elec.fr](mailto:support@mon-club-elec.fr)

**Truc d'utilisation : visualiser ce document en mode diaporama dans le visionneur PDF. Navigation avec les flèches HAUT / BAS ou la souris.**

**En mode fenêtre, activer le panneau latéral vous facilitera la navigation dans le document. Bonne lecture !**

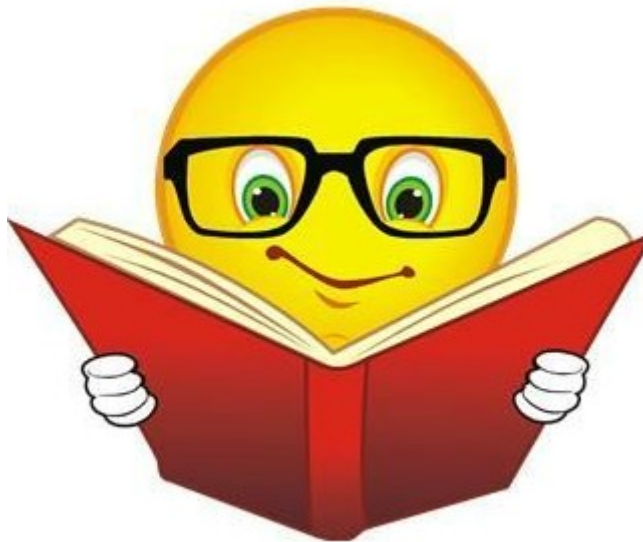
**Lancer également le logiciel Arduino et connecter votre carte Arduino afin de pouvoir tester au fur et à mesure les codes d'exemples !**

## 1. Intro

L'objectif ici est :

- de rappeler le principe de communication du PC [vers](#) la carte Arduino
- d'apprendre à installer une librairie Arduino
- d'utiliser une librairie dédiée pour recevoir une fonction avec paramètres sur le port Série

... afin d'être en mesure d'interagir à l'aide de valeurs reçues sur le port Série avec la carte Arduino à partir du PC.



**Prêt ? C'est parti !**

## 2. Matériel nécessaire pour les ateliers Arduino

Pour cet atelier, vous aurez besoin de tout ou partie des éléments suivants pour pouvoir réaliser les exemples proposés :

### De l'espace de développement Arduino

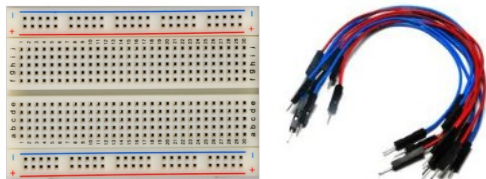


L'espace de développement Arduino associe :

- un ordinateur sous Windows, Mac Os X ou Gnu/Linux (Ubuntu)
- avec le logiciel Arduino installé (voir : <http://www.arduino.cc/>)
- un câble USB
- une carte Arduino UNO ou équivalente.

disponible chez : <http://shop.snootlab.com/> ou <http://www.gotronic.fr/>

### Du nécessaire pour réaliser des montages sans soudure

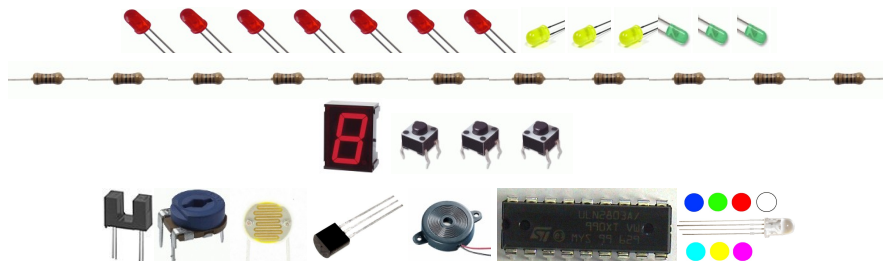


Pour réaliser des montages sans soudure, vous aurez besoin :

- d'une plaque d'essai ou breadboard moyenne (450 points)
- de quelques câbles souples (ou jumpers) mâle/mâle

disponible chez : <http://www.gotronic.fr/>

### De quelques composants de base



**Pour vous simplifier la vie, nous avons négocié ce kit pour vous !**

Vous pouvez commander ce kit complet directement en 1 clic chez notre partenaire

<http://www.gotronic.fr/> avec le code express **701710**

**GO TRONIC**  
ROBOTIQUE ET COMPOSANTS ÉLECTRONIQUES

Pour plus de détails, voir : [http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.ATELIERS](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS)

Pour les ateliers Arduino niveau débutant, vous devrez idéalement disposer des composants suivants :

- des LEDs 5mm Rouges(x20), Vertes (x5) et 3 Jaunes (x5)
- digit à cathode commune rouge 13mm (x1)
- Résistances (1/4w - 5%) de 270 Ohms (x20), 4,7K Ohms (x1), 1K Ohms (x1)
- mini bouton-poussoir (x3)
- Opto-fourche (x 1)
- Résistance variable linéaire 10K (x 1)
- Photo-résistance 7mm (x 1)
- Capteur de température LM35DZ (-55/+150°C - 10mV/°C) (x 1)
- Capsule son piézoélectrique (x 1)
- ULN 2803A (CI amplificateur 8 voies, 500mA/ voie) (x 1)
- LED 5mm multicolore RVB cathode commune (x 1)

### 3. Rappel : Principe de communication de l'Arduino vers le PC

Comme vous le savez déjà, la carte Arduino est (re-)programmable à volonté via le port série USB du PC : c'est ce qui fait toute la simplicité de son utilisation.

Mais la carte Arduino est également capable très simplement de communiquer avec le PC pendant l'exécution d'un programme :

- pour **envoyer des messages vers le PC** (chaîne texte, valeurs numériques) afin de les visualiser sur l'écran sous forme texte ou même sous forme de graphiques (usage avancé) !
- pour **recevoir des messages depuis le PC** (chaîne texte, valeurs numériques) ce qui permettra de contrôler la carte Arduino à partir du clavier ou de la souris par exemple (usage avancé) !

Le langage Arduino dispose de toutes les instructions nécessaires pour réaliser et programmer cette communication au sein d'un programme comme nous allons le voir ici.

La possibilité offerte par le langage Arduino d'afficher des messages sur le PC est l'une des grandes forces de ce système : il est ainsi possible de « voir » de l'intérieur comment un programme fonctionne, ce qui est très puissant pour comprendre, mais aussi mettre au point ses programmes.

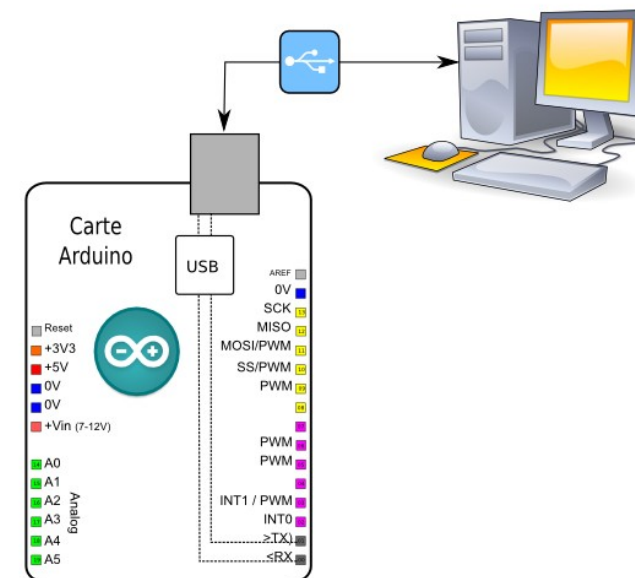
Au final, la carte Arduino programmée pourra fonctionner de 2 façons :

- soit en autonomie, déconnectée du PC une fois programmée,
- soit en communiquant avec le PC pendant l'exécution du programme, réalisant un véritable périphérique USB programmable et personnalisable à souhait !

Programmation par le port USB



Communication par le port USB pendant l'exécution du programme !



## 4. Rappel : Notion de « Classe »

Dans les langages de programmation actuels, les concepteurs ont imaginé la possibilité de pouvoir rassembler des fonctions ensemble lorsqu'elles s'appliquent à une même fonctionnalité.

Par exemple, toutes les fonctions qui s'occupent des opérations mathématiques vont pouvoir être regroupées ensemble.

**A retenir : On appelle « classe » un regroupement de fonctions.**

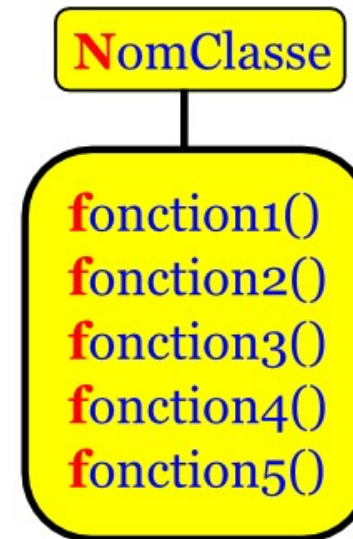
Tout comme une fonction, une classe aura un nom : pour distinguer une classe d'une fonction, **le nom d'une classe commencera par une MAJUSCULE.**

En pratique, lorsque l'on programme en langage Arduino, on n'a pas besoin de créer de classes (ouf !). Mais le langage Arduino ou ses librairies comporte plusieurs classes et il faut donc comprendre ce concept :

- Ainsi, toutes les fonctions qui gèrent la communication avec le port série USB sont rassemblées dans une classe appelée **Serial** : nous allons utiliser cette classe ici.
- la classe LiquidCrystal pour la gestion d'un afficheur LCD
- la classe Servo pour la gestion d'un servomoteur
- etc...

**En pratique, pour utiliser une fonction d'une classe du langage Arduino, on utilisera le nom de la classe + un point + le nom de la fonction.**

*Remarque technique : les instructions de base du langage Arduino, même si elles ne sont pas précédées par un nom de classe, appartiennent toutes à une même classe (implicite) : celle du cœur (ou core) du langage Arduino.*



L'appel d'une fonction d'une classe se fait en séparant le nom de la classe et le nom de la fonction **par un point**

**NomClasse.fonction1()**

## 5. Rappel : la classe Serial

Ainsi, comme on vient de le dire :

**On appelle « classe » un regroupement de fonctions.**

La première classe du langage Arduino que nous avons déjà rencontré est celle qui rassemble toutes les fonctions utilisées pour la communication série USB : cette classe s'appelle **Serial** !

### Les fonctions en « émission » (rappel) :

Les fonctions de la classe Serial sont au nombre d'une dizaine. Nous avons déjà présenté les 3 fonctions qui permettent d'écrire un programme pour afficher des messages vers le PC :

- `begin()` : fonction d'initialisation de la communication USB
- `print()` : fonction d'affichage d'un message sans saut de ligne
- `println()` : fonction d'affichage d'un message avec saut de ligne

### Les fonctions en réception

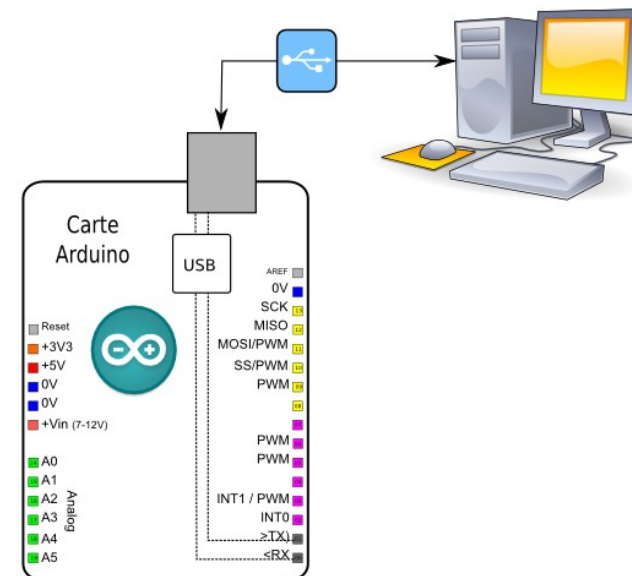
Ici, nous allons découvrir les fonctions de la classe Serial utiles en réception :

- `available()` : renvoie le nombre d'octets présents en réception sur le port Série (Sera donc `true` si caractère présent et `false` sinon...)
- `read()` : lit et renvoie le premier octet entrant en réception sur le port série
- `flush()` : vide la file d'attente en réception du port Série

**Rappel : pour utiliser une fonction d'une classe du langage Arduino, on utilisera le nom de la classe + un point + le nom de la fonction.**

Ces fonctions appartiennent à la classe Serial et nous les utiliserons donc sous la forme : `Serial.available()` ou `Serial.read()` ou `Serial.flush()`

A présent, nous avons tous les éléments pour recevoir des messages en provenance du PC depuis notre carte Arduino !



## 6. Rappel : « Hello world ! » : programme Arduino envoyant un message vers le PC via le port USB

Le principe d'utilisation de la communication USB dans un programme Arduino consiste à :

- initialiser le débit (ou vitesse) de communication une fois pour toute au début du programme (dans la fonction `setup()` )
- utiliser les fonctions `Serial.println()` lorsqu'on en a besoin :
  - soit dans `setup()` pour afficher des messages une seule fois au début du programme,
  - soit dans `loop()` pour afficher des messages à intervalles réguliers ou en boucle.

Notre premier programme Série va :

- initialiser la communication à 115200 bauds avec `Serial.begin()`
- afficher un message toutes les secondes
  - affiche une chaîne : `Serial.println(« mon message »);`
  - pause d'une seconde : `delay(1000);` (**ne pas oublier ! sinon saturation du port USB...** )

**Attention : Une chaîne de caractères s'écrit entre « »**

Une fois le programme écrit :

- le compiler pour vérifier l'absence d'erreur
- **vérifier la carte utilisée (Tools>Board) et le port (Tools>Serial Port)**
- le programmer dans la carte Arduino

Note technique : les chaîne de caractères de message sont stockées en mémoire RAM par défaut, ce qui ne pose pas de problème pour une dizaine de messages. Dès que l'on va écrire beaucoup de messages, il faudra les écrire en mémoire FLASH ce qui se fait depuis Arduino 1.0 par `Serial.print(F(« message »));`

```
void setup() {  
    Serial.begin(115200); // initialise la communication série  
}  
  
void loop() {  
    Serial.println("Hello World !"); // affiche le message  
    delay(1000); // pause de 1 seconde  
}
```



## 7. Rappel : Lancer et paramétrer le Terminal Série pour afficher/recevoir des messages entre le PC et Arduino

Une fois que la carte Arduino est programmée, elle envoie à intervalle régulier des messages vers le PC. Concrètement, vous ne voyez rien se passer à ce stade : **pour voir les messages envoyés par la carte Arduino au PC, vous allez devoir utiliser un logiciel de visualisation.**

Heureusement pour nous, le logiciel Arduino (qui est vraiment très pratique !) dispose d'un tel outil de visualisation : **c'est le Terminal Série**. Pour le lancer :

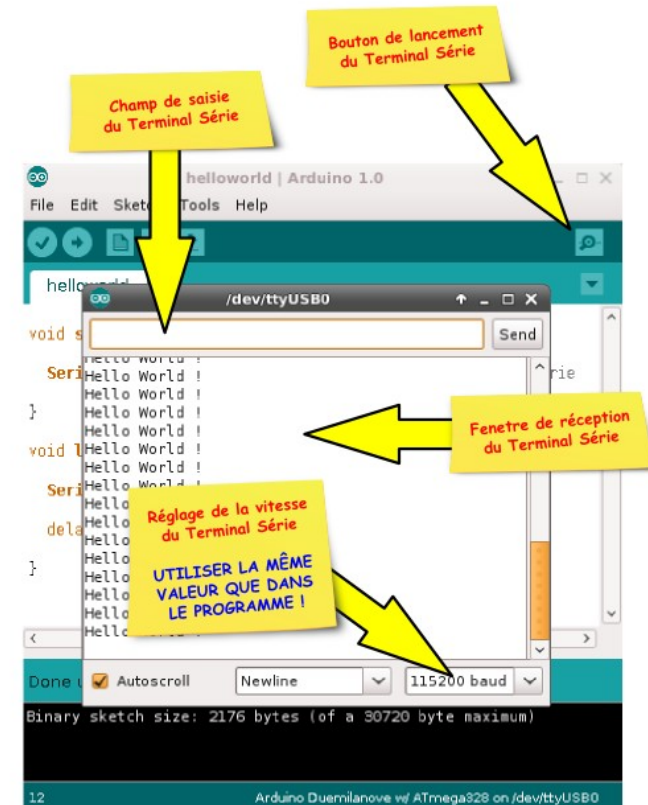
- soit Menu Tool > Serial Monitor
- soit clic sur le bouton Terminal Serie

Une fois la fenêtre du Terminal Série ouverte :

- vérifier que la vitesse de communication est la même que celle que vous avez fixé dans le programme Arduino
- une fois fait vous devriez voir les messages s'afficher dans la fenêtre.

**Bien remarquer que le Terminal Série dispose d'un champ de saisie pour envoyer des caractères vers Arduino : c'est lui que nous utiliser ici pour envoyer des messages vers Arduino.**

- Maintenant que nous avons pu vérifier que le Terminal série fonctionne normalement en « réception », nous allons pouvoir passer à son utilisation en « émission ».





## 8. Rappel : Langage Arduino : Introduction aux bibliothèques

### C'est quoi une bibliothèque Arduino ?

Le langage Arduino comporte de nombreuses instructions comme vous avez pu le constater, une quarantaine en tout. Ces instructions sont intégrées dans ce que l'on appelle le « noyau » ou « cœur » (core en Anglais) du langage Arduino. Ces instructions sont « générales » et servent souvent.

**Le langage Arduino peut cependant être étendu à la demande** avec des instructions dédiées à certaines applications particulières : afin de ne pas surcharger inutilement le « cœur », ces instructions spécifiques ont été intégrées dans des « paquets d'instructions » appelés bibliothèques.

### Comment ça marche ?

Par exemple, si on utilise un afficheur LCD, un servomoteur ou encore si l'on utilise un shield ethernet (réseau), on va intégrer dans notre programme la bibliothèque dédiée correspondante.

### Principe général d'utilisation

Pour intégrer une bibliothèque dans un programme Arduino, c'est très simple : il suffit d'ajouter en début de programme une ligne de la forme :

```
#include <nombibliothèque.h> // bibliothèque pour servomoteur
```

**ATTENTION : l'instruction include est un peu particulière : la ligne commence par un # et il n'y a pas de point virgule de fin de ligne !**

Ensuite, dans le code, au niveau de l'entête déclarative, là où vous déclarez vos variables, il va falloir déclarer un objet (une sorte de super variable) représentant la bibliothèque. Cet objet est en fait une instance (= un exemplaire) d'une Classe (=le moule) qui regroupe les fonctions de la bibliothèque. On a :

```
ClasseObjet monObjet; // déclare un objet
```

Généralement ensuite :

- au niveau de la fonction `setup()`, on initialise l'objet avec les paramètres voulus
- au niveau de la fonction `draw()`, on appelle les fonctions de la bibliothèque sous la forme que vous connaissez déjà :

```
monobjet.fonction( param, param, ..);
```

**Rappel :** pour utiliser une fonction d'une classe du langage Arduino, on utilise le nom de la classe + un point + le nom de la fonction.

Il peut exister des variantes selon les bibliothèques, mais grosso-modo, ça fonctionne de cette façon pour la plupart des bibliothèques Arduino.

### Vous avez déjà utilisé une bibliothèque !

Si vous êtes attentifs à tout ce qu'on a déjà vu, vous me direz que ça ressemble étrangement à l'utilisation de la classe **Serial...** et vous aurez raison ! En fait, la classe Serial est une bibliothèque qui est intégrée implicitement lorsque vous lancez Arduino : c'est pour ça que vous n'avez pas besoin d'utiliser **#include** pour l'utiliser.

### Les bibliothèques standards Arduino

Les bibliothèques Arduino disponibles sont nombreuses, et disposent chacune de quelques fonctions à plusieurs dizaines... ce qui étend considérablement la puissance du langage Arduino et qui en fait aussi tout son intérêt. Voici la liste des bibliothèques standards du langage Arduino ([le logiciel donne la liste...](#)) :

- [La bibliothèque Serial](#) - pour les communications séries entre la carte Arduino et l'ordinateur ou d'autres composants
- [La bibliothèque LCD](#) - pour l'utilisation et le contrôle d'un afficheur LCD alphanumérique standard.
- [La bibliothèque Servo](#) - pour contrôler les servomoteurs.
- [La bibliothèque Stepper](#) - pour contrôler les moteurs pas à pas (nécessite une interface de commande)
- [La bibliothèque Ethernet](#) - pour se connecter à Internet en utilisant le module Arduino Ethernet
- [La bibliothèque EEPROM](#) - référence - pour lire et écrire dans la mémoire EEPROM non volatile.
- [La bibliothèque SD](#) - référence - pour utiliser une carte mémoire SD (utiliser des fichiers, stocker des données, ...)
- [La bibliothèque SoftwareSerial \(Série Logicielle\)](#) - référence - pour communication série logicielle sur n'importe quelles broches de la carte Arduino
- [La bibliothèque Wire / I2C](#) - référence - Interface "deux fils" (TWI/I2C) pour envoyer et recevoir des données sur un réseau de modules ou capteurs.
- [La bibliothèque SPI \(Serial Peripheral Interface\)](#) - pour communication série avec des modules externes supportant le protocole SPI
- [Firmata](#) - pour communiquer avec des applications sur l'ordinateur utilisant un protocole série standard.

**En jaune les plus utiles.** Impressionnant non ? On les étudiera pas à pas...

### Les bibliothèques de la communauté

A côté de ces bibliothèques standards, il existe toute une série de bibliothèques proposées par les uns et les autres et qui concernent des matériels spécifiques, ou autre. Par exemple :

- [La bibliothèque Keypad](#) - pour l'utilisation des claviers matriciels. ([hors référence](#))

Faites un tour ici pour voir ce qui existe : <http://arduino.cc/playground/Main/LibraryList>

## 9. Découvrir et installer les librairies Arduino fournies par la communauté Arduino

### Pour comprendre...

- A côté des librairies Arduino « officielles » installées par défaut avec le langage Arduino, **il existe de nombreuses librairies fournies par la communauté** et qui peuvent être utilisées à la demande en fonction de ses besoins.
- Ces librairies sont variées et ont été créées pour diverses raisons par leurs auteurs :
  - le plus souvent **pour utiliser des matériels précis spécifiques** (par exemple un clavier de PC avec Arduino...)
  - ou **pour apporter des fonctions nouvelles** qui ne sont pas fournies en standard par le langage Arduino (écrire en mémoire Flash, utiliser les interruptions des Timers,...)
  - ou **pour permettre d'utiliser un protocole de communication particulier** avec le langage Arduino (décodage horloge atomique DCF77,...),
  - etc, etc, etc...

Ces librairies ne font pas partie « officiellement » du langage Arduino et par conséquent leur documentation est fournie (ou non !) par l'auteur. Il faut donc parfois mettre un peu les « mains dans le camboui » pour comprendre comment ça marche, mais rien de très sorcier le plus souvent... Des exemples sont presque toujours disponibles.



### Où trouver des librairies Arduino de la communauté ?

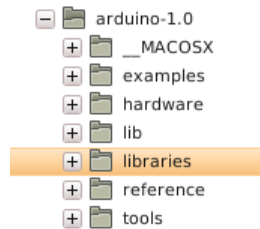
- La plupart des librairies Arduino ont été rassemblées sur le site Arduino, dans ce que l'on appelle le « playground » ou « terrain de jeu ». Allez y faire un tour, vous serez impressionné de la liste existante...
- C'est ici : <http://arduino.cc/playground/Main/LibraryList>

Il y a évidemment « à prendre et à laisser »... Lorsque des librairies de la communauté Arduino sont à connaître, nous vous en parlerons...

En résumé : vous cherchez à faire quelque chose avec Arduino ? Quelqu'un l'a probablement déjà fait avant vous et une librairie existe sûrement !

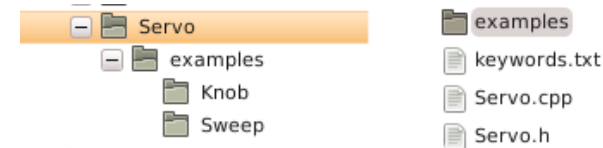
### Où se trouvent les librairies Arduino déjà installées ?

- Les librairies se trouvent dans le répertoire libraries de votre répertoire Arduino :



### Quelle est la structure type d'une librairie Arduino ?

- Une librairie Arduino est écrite en C++ et se présente sous la forme d'un répertoire contenant :
  - 1 ou plusieurs fichiers \*.h
  - 1 ou plusieurs fichiers \*.cpp
  - +/- 1 fichier keywords.txt
  - +/- 1 répertoire exemples



### Comment installer une nouvelle librairie ?

- Télécharger l'archive. au format zip ou autre.
- L'extraire (dans un répertoire si l'archive ne contient pas de répertoire)
- **Vérifier que le nom du répertoire de la librairie est strictement le même que le nom du fichier \*.h ou \*.cpp principal. Corriger au besoin**
- Copier/coller le répertoire de la librairie dans le répertoire libraries de votre répertoire Arduino
- Relancer Arduino et vérifier que la librairie est présente dans le menu **Sketch > ImportLibrary**.
- Puis tester avec un programme d'exemple.

### Envie d'écrire une librairie pour Arduino ?

- C'est un exercice intéressant mais déjà un peu avancé... Cela nécessite d'avoir quelques connaissances en C++
- Pour en savoir plus, c'est par ici : <http://arduino.cc/playground/Code/Library>

## 10. Exemple d'installation et présentation d'une librairie de la communauté : la librairie RunF

### La librairie d'exemple utilisée

- Pour la suite, nous allons utiliser une librairie qui permet de facilement recevoir une fonction avec un ou plusieurs paramètres sur le port série. Cette librairie s'appelle **RunF**.
- Le très gros avantage de cette librairie est de simplifier grandement le code qui serait beaucoup plus lourd pour obtenir le même résultat uniquement avec des fonctions Arduino.

### Télécharger la librairie

- L'archive est disponible ici : <http://code.google.com/p/arduino-runf-library/downloads/list>

### Documentation de la librairie

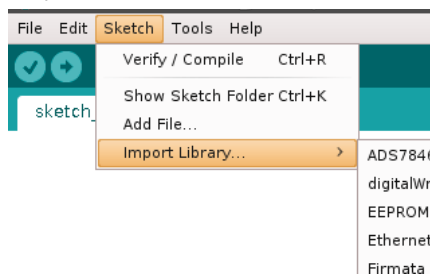
- Un exemple d'utilisation est fourni ici : <http://makerspace56.org/WordPress3/communication-du-pc-vers-arduino/>

### Installation

- Télécharger l'archive. au format zip ou autre. L'extraire
- **Vérifier que le nom du répertoire de la librairie est strictement le même que le nom du fichier \*.h ou \*.cpp principal. Corriger au besoin. Ici le nom est RunF**
- Copier/coller le répertoire de la librairie dans le répertoire libraries de votre répertoire Arduino



- Relancer Arduino et vérifier que la librairie est présente dans le menu **Sketch > ImportLibrary**.



### Le constructeur principal

- Le constructeur principal se nomme RunF et est de la forme :

RunF cmd(listF, listFname);

où :

- **listF** est un tableau de chaînes de caractères définissant le nom des fonctions
- **listFname** est un tableau définissant les fonctions du code associées.

### Fonctions de la librairie

- Une seule fonction dans cette librairie, la fonction **waiting()** qui permet de lire les caractères en réception sur le port série.

### Code d'exemple

```
#include <RunF.h> // inclusion de la librairie

//---- déclaration obligatoire des fonctions à recevoir sur le port Série
typedef void (*typeF)(long*, int); // déclaration de type nécessaire pour la fonction
char *listFname[] = {"maFonction1", "maFonction2", "maFonction3"}; // liste des nom de fonctions
reçues sur le port série nomFonction(param1, param2, .. paramN)
typeF listF[] = {fonction1, fonction2, fonction3}; // liste du nom des fonctions du code
correspondantes

RunF cmd(listF, listFname); // déclaration de l'objet racine RunF utilisé par le programme

void setup() {
    Serial.begin(115200);
}

void loop() {
    cmd.waiting(); // lecture des caractères en réception sur le port série
}

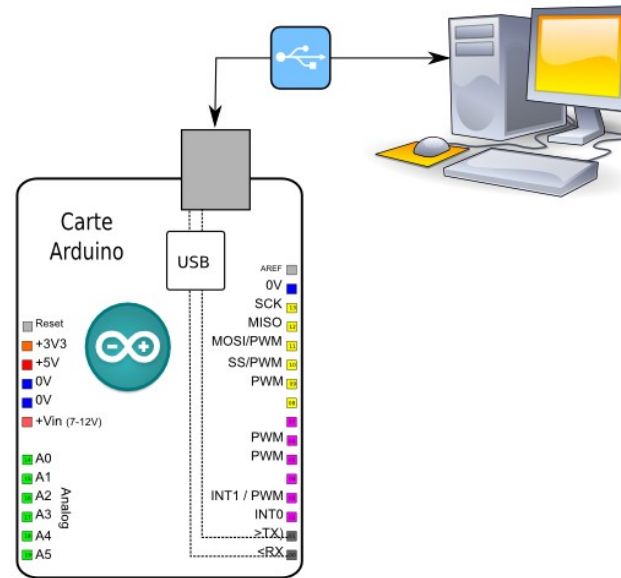
//----- code de la première fonction définie dans le tableau listF[]
void fonction1(long *vars, int qt){
    Serial.println("maFonction1 : ");
    for(int i; i<qt; i++){
        Serial.println(vars[i]);
    }
}

//----- code de la seconde fonction définie dans le tableau listF[]
void fonction2(long *vars, int qt){
    Serial.println("maFonction2 : ");
    for(int i; i<qt; i++){
        Serial.println(vars[i]);
    }
}

//----- code de la troisième fonction définie dans le tableau listF[]
void fonction3(long *vars, int qt){
    Serial.println("maFonction3 : ");
    for(int i; i<qt; i++){
        Serial.println(vars[i]);
    }
}
```

## 11. Réception d'une fonction avec paramètres numériques sur le port Série : le montage

- Le montage est très simple et consiste à garder la carte Arduino connectée sur le port USB : un montage que vous connaissez bien maintenant !



## 12. Réception d'une fonction avec 1 paramètre numérique sur le port Série : le programme

### Ce qu'on va faire ici...

- A présent, on va donc écrire un premier programme qui va recevoir une fonction avec un paramètre numérique entier sur le port série.
- On va logiquement utiliser ici la librairie RunF que nous avons présenté juste avant... Vous devez donc l'avoir installée dans votre répertoire Arduino pour que le programme fonctionne !
- Pour que les choses soient bien claires, je précise que :
  - le programme va reconnaître la fonction sous la forme **envoi()** reçue sur le port série
  - elle devra être écrite sous la forme **envoi(valeur)** sur le port série pour être reconnue par le programme
  - la réception de cette fonction sur le port série appellera une fonction au sein du programme appelée **codeEnvoi()** contenant le code à exécuter lors d'une réception valide sur le port série.

### Entête déclarative

#### Inclusion de la librairie RunF

- on commence par inclure la librairie **RunF** à l'aide de l'instruction **#include** (attention, pas de ;) )

#### Déclaration des fonctions reconnues par le programme

- Cette partie est essentielle pour le programme et a une syntaxe un peu particulière que **vous n'avez pas besoin de tout comprendre : un copier/coller suffira !** Le code utilisé est du C++
- Le point important est de **définir la chaîne de caractères du nom de la fonction qui sera reçue sur le port série** d'une part (ici « **envoi** »)
- Ainsi que **le nom de la fonction du code associée** d'autre part (ici **codeEnvoi**). **Cette fonction devra exister dans le programme ++ !**

#### Déclaration de l'objet RunF principal

- Pour accéder aux fonctions de la librairie RunF, il faut déclarer un objet de type RunF que l'on appellera ici **receptionSerie**.
- La déclaration de cet objet se base logiquement sur les 2 tableaux de chaînes et de fonctions déclarés juste avant.

```
#include <RunF.h> // inclusion de la librairie

//---- déclaration obligatoire des fonctions à recevoir sur le port Série
typedef void (*typeF)(long*, int); // déclaration de type nécessaire
pour la fonction
char *listFname[] = {"envoi"}; // liste des nom de fonctions reçues sur
le port série nomFonction(param1, param2, ., paramN)
typeF listF[] = {codeEnvoi}; // liste du nom des fonctions du code
correspondantes

RunF receptionSerie(listF, listFname); // déclaration de l'objet racine
RunF utilisé par le programme
```

Remarquer la relative simplicité de la déclaration à effectuer : pas besoin de déclarer d'objet String de réception ni de variables numériques de réception. La seule chose à faire est de définir le nom de la fonction à recevoir sur le port série et la fonction du programme qui lui sera associée !

## Fonction **setup()**

- La fonction **setup()** est très simple : on initialise la communication série avec l'instruction **Serial.begin(vitesse)**. On utilisera 115200 bauds.

```
void setup() {  
    Serial.begin(115200); // initialise port série  
    // vérifier que le débit est le meme dans le Terminal Serie  
    // IMPORTANT : sélectionner l'option "No Line Ending" dans le Terminal  
    Serie !  
}  
// fin setup
```

## Fonction **loop()**

- On « écoute » le port série à l'aide de la fonction **waiting()** de la librairie. Par défaut, cette fonction n'attend pas de saut de ligne de fin de chaîne.

La chaîne attendue en réception est de la forme :  
nomFonction(param1, param2, ... , paramN)

```
void loop() {  
    receptionSerie.waiting(); // lecture des caractères en réception sur  
    le port série  
}  
// fin loop
```

### La fonction **loop()** qui se résume à une ligne de code !! Qui dit mieux ?

Rappelez-vous du code utilisé pour simplement recevoir une chaîne de caractère String sur le port série... Une bonne dizaine de lignes étaient nécessaires, avec l'utilisation d'une fonction **while()**, etc... Vous voyez dès lors tout l'intérêt de la librairie RunF !





## Fonction codeEnvoi()

- Cette fonction doit avoir le même nom que celui déclaré en début de programme au niveau du tableau listF[] !
- Cette fonction sera appelée lorsqu'une chaîne de la forme envoi(xxx,yyy,zzz) sera reçue sur le port série et le code qu'elle contient sera exécuté.
- Cette fonction reçoit en paramètres (obligatoire) :
  - le tableau des valeurs des paramètres (vars)
  - une variable précisant le nombre de paramètres reçus (qt)
- Ici, le code de la fonction :
  - affiche la fonction reçue sur le port série,
  - affiche les paramètres reçus
  - analyse rapidement le nombre de paramètres et affiche des messages en conséquence !

Le code de cette fonction peut être très simple si on le souhaite...

Ici, je prends le temps de détailler l'accès aux paramètres reçus par la fonction sur le port Série car c'est ce point qui va être le plus utile pour la suite. Et donc, le code est légèrement plus étoffé, mais rien d'inaccessible !

```
//----- code de la fonction définie dans le tableau listF[]
void codeEnvoi(long *vars, int qt) { // vars est le tableau de
paramètres et qt leur nombre

    //----- affiche la fonction reçue au format envoi(val1, val2)
    Serial.print("envoi("); // pour affichage envoi(val1, val2)
    if (qt==0) Serial.println(")");

    for(int i; i<qt; i++){ // défile les paramètres
        Serial.print(vars[i]);
        if (i<qt-1) Serial.print(","); else Serial.println(")"); // pour
affichage envoi(val1, val2)
    } // fin for

    //----- message d'analyse de la fonction reçue -----
    if (qt>1) Serial.println ("Fonction non valide : trop de
parametres !");
    if (qt==0) Serial.println ("Fonction non valide : pas assez de
parametres !");

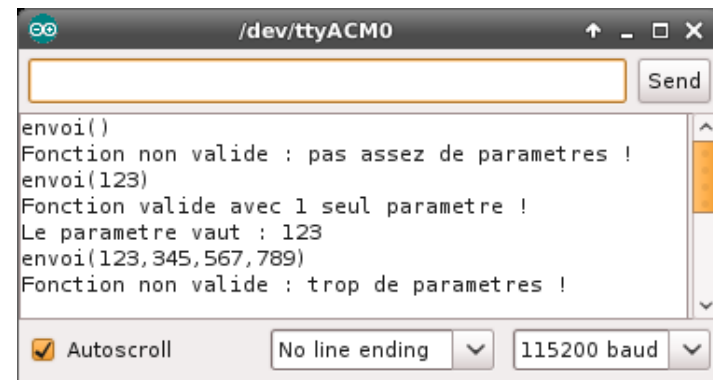
    if (qt==1) { // si 1 seul parametre reçu

        Serial.println ("Fonction valide avec 1 seul parametre !");
        Serial.print ("Le parametre vaut : ");
        Serial.println (vars[0]); // la première valeur a l'indice 0 !

    } // fin if qt==1
} // --- fin codeEnvoi
```

## Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série (Tools > Serial Monitor)
- fixer le débit à la même valeur que celle utilisée pour l'instruction **Serial.begin**(vitesse). Ici, 115200 bauds.
- Régler également les paramètres de transmission de la chaîne de caractère à l'aide de la 2ème liste défilante. Mettre sur « No Line Ending » pour aucun ajout après la chaîne saisie.
- A présent, saisir des chaînes sous la forme envoi(), puis envoi(123) puis envoi(123,456,789), etc... Observer les messages obtenus



### 13. Réception d'une fonction avec plusieurs paramètres numériques entiers sur le port Série : le programme

#### Ce qu'on va faire ici...

- Allez, on continue... ! On va maintenant écrire un programme qui va recevoir une fonction avec plusieurs paramètres numériques entiers sur le port série.



En fait, je dois l'avouer, je me moque un peu de vous... ! Ce programme est en fait exactement le même que précédemment ! **Toute la puissance de la librairie RunF apparaît ici : que vous receviez aucun paramètre, un paramètre... ou 12 paramètres, le code est le même !!** La seule chose qui change est le code de la fonction appelée lors de la réception d'une chaîne valide.

- Donc, je reprécise à nouveau que :
  - le programme va reconnaître la fonction sous la forme **envoi()** reçue sur le port série
  - elle devra être écrite sous la forme **envoi(valeur1, valeur2, valeur3, ... , valeurN)** sur le port série pour être reconnue par le programme
  - la réception de cette fonction sur le port série appellera une fonction au sein du programme appelée **codeEnvoi()** contenant le code à exécuter lors d'une réception valide sur le port série.

#### Entête déclarative

##### Inclusion de la librairie RunF

- on commence par inclure la librairie **RunF** à l'aide de l'instruction `#include` (attention, pas de ;) )

##### Déclaration des fonctions reconnues par le programme

- Cette partie est essentielle pour le programme et a une syntaxe un peu particulière que vous n'avez pas besoin de comprendre : un copier/coller suffira ! Le code utilisé est du C++
- Le point important est de définir la chaîne de caractères du nom de la fonction qui sera reçue sur le port série d'une part (ici « **envoi** »)
- Ainsi que le nom de la fonction du code associée d'autre part (ici **codeEnvoi**). Cette fonction devra exister dans le programme ++ !

##### Déclaration de l'objet RunF principal

- Pour accéder aux fonctions de la librairie RunF, il faut déclarer un objet de type RunF que l'on appellera ici `receptionSerie`.
- La déclaration de cet objet se base logiquement sur les 2 tableaux de chaînes et de fonctions déclarés juste avant.

```
#include <RunF.h> // inclusion de la librairie

//---- déclaration obligatoire des fonctions à recevoir sur le port Série
typedef void (*typeF)(long*, int); // déclaration de type nécessaire pour la fonction
char *listFname[] = {"envoi"}; // liste des nom de fonctions reçues sur le port série
nomFonction(param1, param2, ., paramN)
typeF listF[] = {codeEnvoi}; // liste du nom des fonctions du code correspondantes

RunF receptionSerie(listF, listFname); // déclaration de l'objet racine RunF utilisé par le programme
```

## Fonction **setup()**

- La fonction **setup()** est très simple : on initialise la communication série avec l'instruction **Serial.begin(vitesse)**. On utilisera 115200 bauds.

```
void setup() {  
    Serial.begin(115200); // initialise port série  
    // vérifier que le débit est le meme dans le Terminal Serie  
    // IMPORTANT : sélectionner l'option "No Line Ending" dans le Terminal  
    Serie !  
}  
// fin setup
```

## Fonction **loop()**

- On « écoute » le port série à l'aide de la fonction **waiting()** de la librairie. Par défaut, cette fonction n'attend pas de saut de ligne de fin de chaîne.

La chaîne attendue en réception est de la forme :  
nomFonction(param1, param2, ... , paramN)

```
void loop() {  
    receptionSerie.waiting(); // lecture des caractères en réception sur  
    le port série  
}  
// fin loop
```

**La fonction loop() qui se résume toujours qu'à une ligne de code !! Qui dit mieux ?**



## Fonction `codeEnvoi()`

- Cette fonction doit avoir le même nom que celui déclaré en début de programme au niveau du tableau `listF[]` !
- Cette fonction sera appelée lorsqu'une chaîne de la forme `envoi(xxx,yyy,zzz)` sera reçue sur le port série et le code qu'elle contient sera exécuté.
- Cette fonction reçoit en paramètres (obligatoire) :
  - le tableau des valeurs des paramètres (`vars`)
  - une variable précisant le nombre de paramètres reçus (`qt`)
- Ici, le code de la fonction affiche tous les paramètres reçus !

Une simple boucle pour accéder à tous les paramètres sans avoir besoin de fixer à l'avance le nombre de paramètres... mais tout en étant en mesure d'analyser les paramètres reçus, leur nombre, etc...

Très fort !

```
//----- code de la fonction définie dans le tableau listF[]
void codeEnvoi(long *vars, int qt) { // vars est le tableau de
paramètres et qt leur nombre

    //----- affiche les paramètres de la fonction reçue

    Serial.println("Fonction envoi() avec les paramètres suivants :"); //
pour affichage envoi(val1, val2)

    if (qt==0) Serial.println ("Aucun parametre !");
    if (qt==0) Serial.println ("Aucun parametre !");

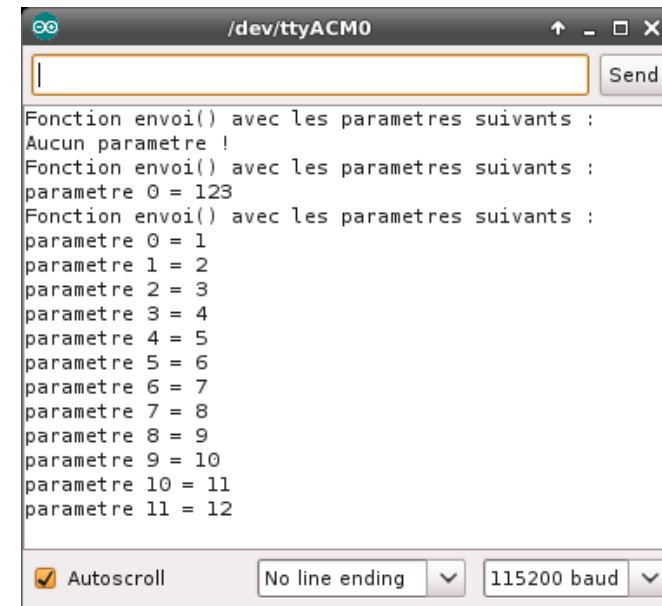
    for(int i=0; i<qt; i++){ // défile les paramètres
        Serial.print("parametre ");
        Serial.print(i);
        Serial.print(" = ");
        Serial.println(vars[i]);

    } // fin for

} // --- fin codeEnvoi
```

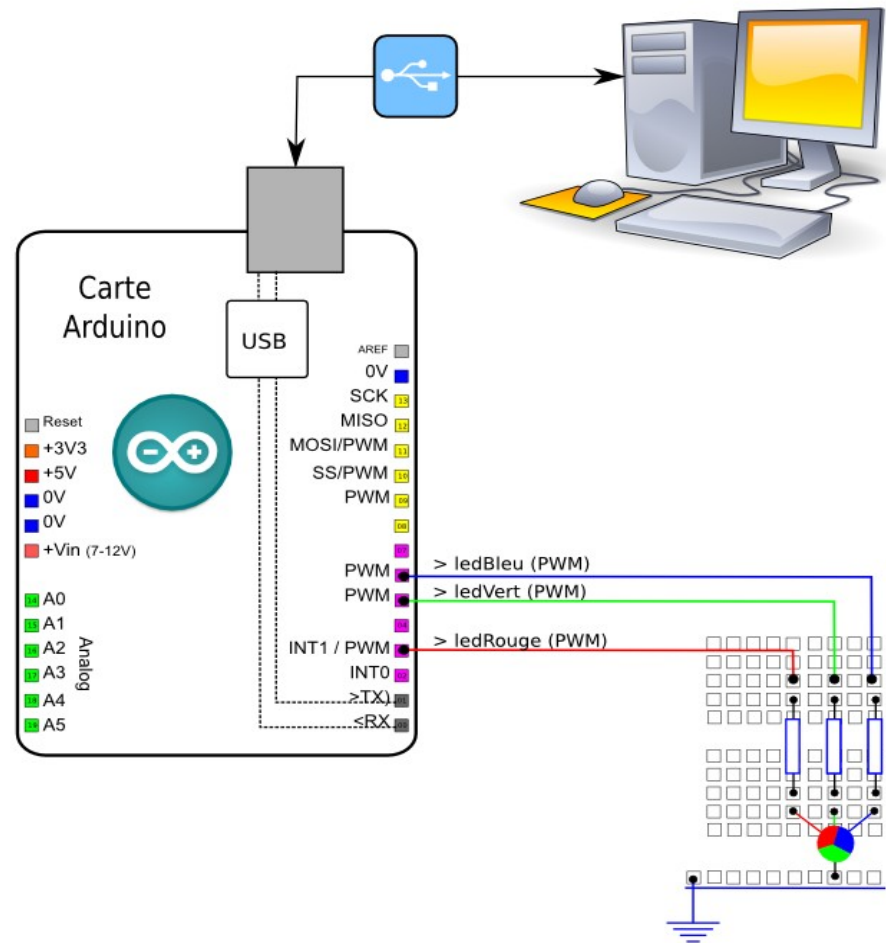
## Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série (Tools > Serial Monitor)
- fixer le débit à la même valeur que celle utilisée pour l'instruction `Serial.begin(vitesse)`. Ici, 115200 bauds.
- Régler également les paramètres de transmission de la chaîne de caractère à l'aide de la 2ème liste défilante. Mettre sur « No Line Ending » pour aucun ajout après la chaîne saisie.
- A présent, saisir des chaînes sous la forme `envoi()`, puis `envoi(123)` puis `envoi(123,456,789)`, etc... Observer les messages obtenus !
- On peut passer jusqu'à 12 paramètres sans aucun problème !



## 14. Contrôler une LED multicolore RGB par le port série : le montage

Dans ce programme, nous allons utiliser une LED RGB à cathode commune connectée sur 3 broches PWM. La carte Arduino communiquera avec le PC via le port série.



Pour plus de détails, voir l'atelier dédié aux « sorties analogiques : son et PWM »

## 15. Contrôler une LED multicolore RGB par le port série : le programme

### Ce que l'on va faire ici...

- Ici, nous allons fixer les couleurs d'une LED RGB à l'aide d'une fonction à 3 paramètres passée sur le port Série, soit sous la forme RGB(xxx,yyy,zzz) soit sous la forme R(xxx), G(yyy) ou B(zzz) où les 3 valeurs représentent respectivement le dosage du rouge, du vert et du bleu.
- Nous utiliserons ici la librairie RunF pour implémenter la réception de fonctions avec paramètres sur le port série.

### Entête déclarative

#### Inclusion des librairies

- on commence par inclure la librairie **RunF** à l'aide de l'instruction `#include` (**attention, pas de ;**)

#### Déclarations utiles pour la LED RGB

- En ce qui concerne la LED RGBs, on déclare :
  - les 3 broches de couleur de la LED

#### Déclaration des fonctions reconnues par le programme

- Cette partie est essentielle pour le programme et a une syntaxe un peu particulière que vous n'avez pas besoin de comprendre : un copier/coller suffira ! Le code utilisé est du C++
- Le point important est :
  - de définir les chaînes de caractères du noms des fonctions qui seront reçues sur le port série d'une part (ici 4 chaînes !)
  - Ainsi que les noms des fonctions du code associée d'autre part (ici 4 fonctions). Ces fonctions devront exister dans le programme !

#### Déclaration de l'objet RunF principal

- Pour accéder aux fonctions de la librairie RunF, il faut déclarer un objet de type RunF que l'on appellera ici `receptionSerie`.
- La déclaration de cet objet se base logiquement sur les 2 tableaux de chaînes et de fonctions déclarés juste avant.

```
//--- entete déclarative

//--- inclusion des librairies

#include <RunF.h> // inclusion de la librairie
// disponible ici : http://code.google.com/p/arduino-runf-
library/downloads/list
// article ici : http://makerspace56.org/WordPress3/communication-du-pc-
vers-arduino/

//--- constantes des broches utilisées
const int ledR=3; // constante désignant la broche de la LED
const int ledV=5; // constante désignant la broche de la LED
const int ledB=6; // constante désignant la broche de la LED

//----- constantes utiles -----
const int R=0; //--- constante binaire couleur R
const int V=0; //--- constante binaire couleur G
const int B=0; //--- constante binaire couleur B

//---- déclaration obligatoire des fonctions à recevoir sur le port Série
typedef void (*typeF)(long*, int); // déclaration de type nécessaire
pour la fonction
char *listFname[] = {"R", "V", "B", "RVB", "init"};
// liste des nom de fonctions reçues sur le port série nomFonction(param
1, param2, .. paramN)
typeF listF[] = {codeR, codeV, codeB, codeRVB, codeInit};
// liste du nom des fonctions du code correspondantes

RunF receptionSerie(listF, listFname); // déclaration de l'objet racine
RunF utilisé par le programme
```



## Fonction **setup()**

### *Initialisation de la communication série*

- on initialise la communication série avec l'instruction **Serial.begin**(vitesse). On utilisera 115200 bauds.

### *Messages d'initialisation*

- on affiche la liste des chaînes reconnues par le programme ce qui facilitera l'utilisation

```
//--- la fonction setup() : exécutée au début et 1 seule fois

void setup() {

    //----- configuration des broches utilisées -----
    pinMode(ledR, OUTPUT); // met la broche en sortie
    pinMode(ledV, OUTPUT); // met la broche en sortie
    pinMode(ledB, OUTPUT); // met la broche en sortie

    //----- Initialisation Série -----
    Serial.begin(115200); // initialise la vitesse de la connexion série
    //-- utilise la meme vitesse dans le Terminal Série
    // IMPORTANT : sélectionner l'option "No Line Ending" dans le
    Terminal Serie !

    //-----
    Serial.println("--- Instructions reconnues: ---");
    Serial.println("R(rouge)");
    Serial.println("V(vert)");
    Serial.println("B(bleu)");
    Serial.println("RVB(rouge, vert, bleu)");
    Serial.println("Utiliser des valeurs 0 - 255 ");
    Serial.println("-----");

} // fin de la fonction setup()
```

## Fonction **loop()**

### *Gestion du port Série*

- La gestion de la réception sur le port série se résume à l'appel de la fonction **waiting()** de la librairie RunF.

```
//--- la fonction loop() : exécutée en boucle sans fin
void loop() {

    receptionSerie.waiting(); // lecture des caractères en réception sur
    le port série

} // fin de la fonction loop()
```

## Fonction gérant les couleurs de la LED RGB

Cette fonction :

- reçoit en paramètres la valeur à utiliser pour chacune des 3 couleurs
- met l'impulsion voulue sur la broche correspondante de la LED RGB à l'aide de l'instruction `analogWrite()`

```
//---- fonction pour combiner couleurs ----  
  
void ledRVB(int Rouge, int Vert, int Bleu) {  
  
    analogWrite(ledR,Rouge); // proportion de rouge  
    analogWrite(ledV,Vert); // proportion de vert  
    analogWrite(ledB,Bleu); // proportion de bleu  
  
}
```

## Fonctions gérant fonctions reçues sur le port Série

- Cette partie du code est un peu plus longue, mais ne comporte aucun élément compliqué.
- Pour chaque fonction, on reçoit le tableau de valeurs long (les paramètres) et une valeur int représentant le nombre des paramètres
- Ensuite, on accède aux valeurs des paramètres sous la forme vars[index] tel que l'index=0 correspond au premier élément.
- Pour chaque fonction reconnue, le code voulu est exécuté.
- Des messages sont affichés pour améliorer le suivi de l'exécution du programme.

**Remarquer la facilité avec laquelle on peut déclarer des fonctions qui seront attachées aux chaînes reçues sur le port Série, avec plusieurs paramètres numériques au besoin.**

```
//----- fonctions reçues sur le port série -----  
  
//---- fonction codeR  
void codeR(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre  
  
    if (qt==1) { // si 1 paramètre reçu  
        Serial.print("Valeur du rouge = ");  
        Serial.println(vars[0]);  
  
        analogWrite(ledR,vars[0]); // proportion de rouge  
  
    }  
    else Serial.println("Fonction non valide ! ");  
}  
// --- fin codeR  
  
//---- fonction codeV  
void codeV(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre  
  
    if (qt==1) { // si 1 paramètre reçu  
        Serial.print("Valeur du vert = ");  
        Serial.println(vars[0]);  
  
        analogWrite(ledV,vars[0]); // proportion de rouge  
  
    }  
    else Serial.println("Fonction non valide ! ");  
}  
// --- fin codeV  
  
//---- fonction codeB  
void codeB(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre  
  
    if (qt==1) { // si 1 paramètre reçu  
        Serial.print("Valeur du bleu = ");  
        Serial.println(vars[0]);
```

```

    analogWrite(ledB, vars[0]); // proportion de rouge
}
else Serial.println("Fonction non valide ! ");
} // --- fin codeB

//---- fonction codeRGB
void codeRVB(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre

    if (qt==3) { // si 1 paramètre reçu
        Serial.print("Valeur du rouge = ");
        Serial.print(vars[0]);

        Serial.print("Valeur du vert = ");
        Serial.print(vars[1]);

        Serial.print("Valeur du bleu = ");
        Serial.println(vars[2]);

        ledRVB(vars[0], vars[1], vars[2]); // appelle la fonction ledRVB
    }
    else Serial.println("Fonction non valide ! ");
} // --- fin codeRGB

//----- fonction init()
void codeInit(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre

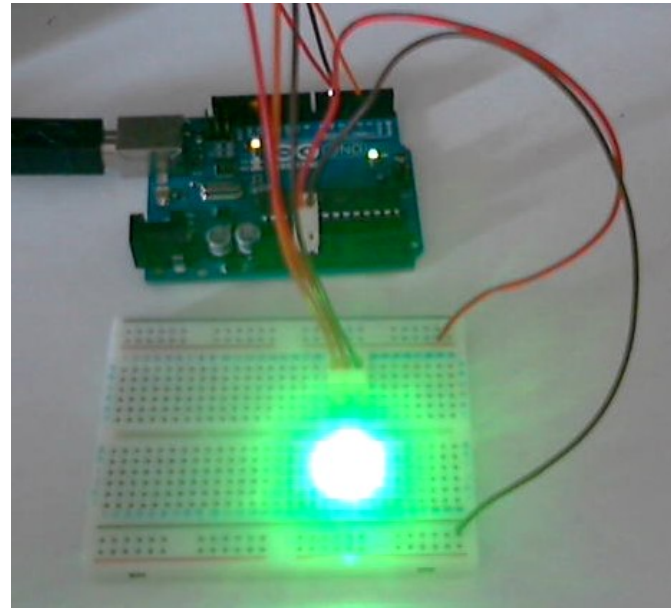
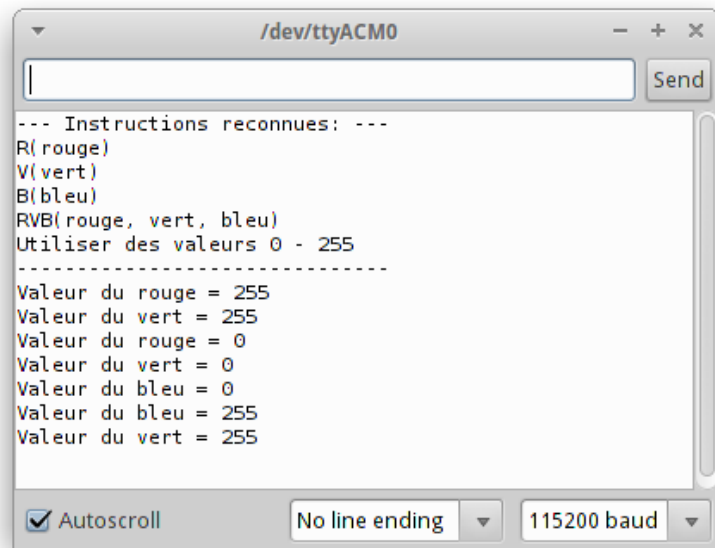
    if (qt==0) { // si aucun paramètre reçu
        Serial.println("Initialisation de la LED : toutes les couleurs =0 !");

        ledRVB(0, 0, 0); // appelle la fonction ledRVB
    }
    else Serial.println("Fonction non valide ! ");
}
}

```

## Fonctionnement du programme

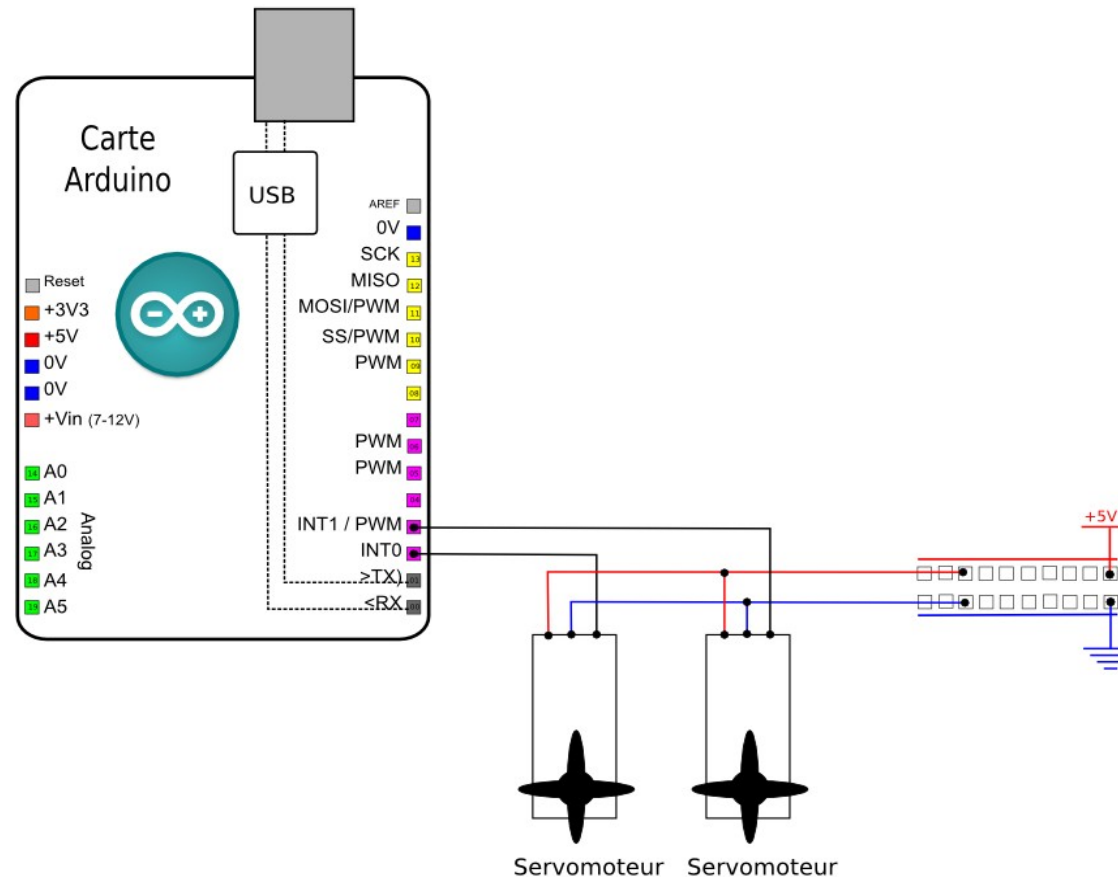
- Ouvrir le Terminal Série (Tools > Serial Monitor) et fixer le débit à la même valeur que celle utilisée pour l'instruction `Serial.begin(vitesse)`. Ici, **115200** bauds.
- Régler également les paramètres de transmission de la chaîne de caractère à l'aide de la 2ème liste défilante. **Mettre sur « New Line »** pour ajout du « saut de ligne » après la chaîne saisie.
- saisir l'une des chaînes reconnues en mettant entre parenthèses les valeurs souhaitées : les moteurs réalisent l'action demandée ! Très très pratique pour tester son petit robot par le port Série de manière très fine.
- Noter la possibilité d'étalonner les servomoteurs au besoin avec ce programme avec les instructions `testServoD()` et `testServoG()`



Les fonctions reçues sur le port Série contrôlent les couleurs de la LED RGB !

## 16. Contrôle de la vitesse et du sens de 2 servomoteurs à rotation continue par le port Série : le montage

- A présent, passons à l'utilisation concrète de cette fonctionnalité ! Nous allons reprendre ici le montage utilisant 2 servomoteurs à rotation continue (voir l'atelier consacré aux servomoteurs à rotation continue pour plus de détails) : Le montage est relativement facile, même si les choses se compliquent un peu.
- Pour chaque servomoteur, on connecte :
  - le + (**fil rouge**) et le – (**fil noir ou marron**) du servomoteur respectivement au **+5V** et au **0V** de la carte Arduino
  - le **fil de commande** (fil blanc ou orange) à une **broche numérique en sortie** de la carte Arduino.





## 17. Contrôle de la vitesse et du sens de 2 servomoteurs à rotation continue par le port Série : le programme

Ce que l'on va faire ici...

- Un des objectifs premiers à l'utilisation des servomoteurs à rotation continue est la réalisation d'un robot mobile d'initiation. Avec 2 servomoteurs à rotation continue, il est en effet assez facile de contrôler le sens de rotation et la vitesse pour chaque moteur et donc de programmer des actions synchronisées entre les 2 servomoteurs, notamment la marche avant, la marche arrière, la bifurcation vers la droite ou vers la gauche et bien sûr l'arrêt !
- Le tout, rappelons-le, avec seulement 2 broches de la carte Arduino et aucune interface moteur !! Dans ce programme, je vous propose de poser les bases du contrôle d'un tel robot à partir du Terminal Série. Non, non, vous ne rêvez pas... !

### Entête déclarative

#### Inclusion des bibliothèques

- On commence par inclure la bibliothèque **Servo**
- on commence par inclure la bibliothèque **RunF** à l'aide de l'instruction **#include** (attention, pas de ;) )

#### Déclaration des fonctions reconnues par le programme

- Cette partie est essentielle pour le programme et a une syntaxe un peu particulière que vous n'avez pas besoin de comprendre : un copier/coller suffira ! Le code utilisé est du C++
- Le point important est :
  - de définir les chaînes de caractères des noms des fonctions qui seront reçues sur le port série d'une part (ici 11 chaînes !)
  - Ainsi que les noms des fonctions du code associée d'autre part (ici 11 fonctions). Ces fonctions devront exister dans le programme !

#### Déclaration de l'objet RunF principal

- Pour accéder aux fonctions de la bibliothèque RunF, il faut déclarer un objet de type RunF que l'on appellera ici receptionSerie.
- La déclaration de cet objet se base logiquement sur les 2 tableaux de chaînes et de fonctions déclarés juste avant.

#### Déclarations utiles pour les servomoteurs

- En ce qui concerne les servomoteurs, on déclare :
  - 2 constantes pour désigner les moteurs droit et gauche,
  - les constantes de la valeur neutre et des vitesses maximales sous forme d'un tableau (valeurs inversées pour chaque servomoteur)
  - un tableau de constantes des broches pour les servomoteurs
- On déclare également un tableau de 2 objets **Servo**

```
//---- inclusion des bibliothèques
#include <Servo.h> // inclut la bibliothèque Servo

#include <RunF.h> // inclusion de la bibliothèque
// disponible ici : http://code.google.com/p/arduino-runf-library/downloads/list
// article ici : http://makerspace56.org/WordPress3/communication-du-pc-vers-arduino/

//---- déclaration obligatoire des fonctions à recevoir sur le port Série
typedef void (*typeF)(long*, int); // déclaration de type nécessaire pour la fonction
char *listFname[] = {"stop", "avantD", "avantG", "arriereD", "arriereG", "enAvant", "enArriere", "tourneD", "tourneG", "testServoD", "testServoG"};
// liste des nom de fonctions reçues sur le port série nomFonction(param1, param2, .., paramN)
typeF listF[] = {codeStop, codeAvantD, codeAvantG, codeArriereD, codeArriereG, codeEnAvant, codeEnArriere, codeTourneD, codeTourneG, testServoD, testServoG};
// liste du nom des fonctions du code correspondantes

RunF receptionSerie(listF, listFname); // déclaration de l'objet racine RunF utilisé par le programme

//--- variables et constantes pour les servomoteurs
const int Droit=0; // servo Droit a l'indice 0
const int Gauche=1; // servo Droit a l'indice 1

/* futaba RC
const int neutre=1480; // largeur impulsion arrêt
const int maxAV[2]={1230,1730}; // largeur impulsion vitesse max en avant
const int maxAR[2]={1730,1230}; // largeur impulsion vitesse max en arriere
*/

const int neutre=1430; // largeur impulsion arrêt
const int maxAV[2]={1130,1730}; // largeur impulsion vitesse max en avant
const int maxAR[2]={1730,1130}; // largeur impulsion vitesse max en arriere

//const int brocheServo[2]={2,3}; // broches des servomoteur
const int brocheServo[2]={14,15}; // broches des servomoteur - CD Simple Bot

//int impulsServo=0; // largeur impulsion servomoteur en usecondes

Servo servo[2]; // déclaration d'un objet servomoteur
```

## Fonction **setup()**

### **Initialisation de la communication série**

- on initialise la communication série avec l'instruction **Serial.begin(vitesse)**. On utilisera 115200 bauds.

### **Messages d'initialisation**

- on affiche la liste des chaînes reconnues par le programme ce qui facilitera l'utilisation

```
//--- la fonction setup() : exécutée au début et 1 seule fois
void setup() {

    Serial.begin(115200); // initialise la vitesse de la connexion série
    //-- utilise la meme vitesse dans le Terminal Série
    // IMPORTANT : sélectionner l'option "No Line Ending" dans le
    Terminal Serie !

    Serial.println("--- Instructions reconnues: ---");
    Serial.println("stop()");
    Serial.println("avantD() | avantD(vitesse%)");
    Serial.println("avantG() | avantG(vitesse%)");
    Serial.println("arriereD() | arriereD(vitesse%)");
    Serial.println("arriereG() | arriereG(vitesse%)");
    Serial.println("enAvant() | enAvant(vitesse%)");
    Serial.println("enArriere() | enArriere(vitesse%)");
    Serial.println("tourneD() | tourneD(vitesse%)");
    Serial.println("tourneG() | tourneG(vitesse%)");
    Serial.println("testServoD(largeurmicros)");
    Serial.println("testServoG(largeurmicros)");
    Serial.println("-----");

} // fin de la fonction setup()
```

## Fonction **loop()**

### **Gestion du port Série**

- La gestion de la réception sur le port série se résume à l'appel de la fonction **waiting()** de la librairie RunF.

```
//--- la fonction loop() : exécutée en boucle sans fin
void loop() {

    receptionSerie.waiting(); // lecture des caractères en réception sur
    le port série

} // fin de la fonction loop()
```

## Fonctions gérant les mouvements individuels des servomoteurs

### Fonction de gestion du sens de rotation avant

- Cette fonction reçoit l'indice du servomoteur à utiliser ainsi que la vitesse à utiliser entre 0 et 100%
- Cette fonction ne renvoie rien (type void)
- Le code de la fonction consiste à :
  - attacher le servomoteur à la broche correspondante
  - à calculer la largeur de l'impulsion à utiliser à l'aide de la fonction `map()` qui va ré-échelonner la valeur en pourcentage en une valeur proportionnelle en microsecondes comprise entre la valeur neutre et la valeur de la vitesse maximale positive.
  - on applique cette largeur d'impulsion au servomoteur à l'aide de la fonction `writeMicroseconds()`
  - Une courte pause est appliquée.

### Fonction de gestion du sens de rotation arrière

- Fonction identique avec comme différence l'utilisation de la valeur de la vitesse maximale arrière au lieu de la vitesse maximale avant au niveau de la fonction `map()`

```
//----- fonctions des mouvements de base des servomoteurs -----  
  
//----- fonction de controle de la vitesse avant d'un servomoteur  
void servoAvant(int indiceServo, int vitesseIn) {  
  // -- vitesse entre 10 et 100% --  
  
    int impulsServo; // variable locale  
  
    servo[indiceServo].attach(brocheServo[indiceServo]); // attache le  
servomoteur à la broche  
  
    impulsServo=map(vitesseIn, 0, 100, neutre, maxAV[indiceServo]); //  
calcul largeur impulsion AV correspondante  
    servo[indiceServo].writeMicroseconds(impulsServo);  
    delay(10); // pause  
  
} // fin servoAvant  
  
//----- fonction de controle de la vitesse arriere d'un servomoteur  
void servoArriere(int indiceServo, int vitesseIn) {  
  // -- vitesse entre 10 et 100% --  
  
    int impulsServo; // variable locale  
  
    servo[indiceServo].attach(brocheServo[indiceServo]); // attache le  
servomoteur à la broche  
  
    impulsServo=map(vitesseIn, 0, 100, neutre, maxAR[indiceServo]); //  
calcul largeur impulsion AV correspondante  
    servo[indiceServo].writeMicroseconds(impulsServo);  
    delay(10); // pause  
  
} // fin servoArriere
```

## Fonctions gérant les mouvements synchronisés des servomoteurs

### Fonction gérant la marche avant

- Cette ne renvoie rien (type void)
- Cette fonction reçoit en paramètre la vitesse (entre 0 et 100%)
- Le code de cette fonction consiste à mettre en marche avant les 2 servomoteurs en se basant sur la fonction de gestion individuelle de la marche avant.

### Fonction gérant la marche arrière

- Cette ne renvoie rien (type void)
- Cette fonction reçoit en paramètre la vitesse (entre 0 et 100%)
- Le code de cette fonction consiste à mettre en marche arrière les 2 servomoteurs en se basant sur la fonction de gestion individuelle de la marche arrière.

### Fonction gérant la bifurcation à droite et à gauche

- Cette ne renvoie rien (type void)
- Cette fonction reçoit en paramètre la vitesse (entre 0 et 100%)
- Le code de cette fonction consiste à mettre en marche arrière l'un des 2 servomoteurs et en marche arrière le second en se basant sur la fonction de gestion individuelle de la marche arrière/avant.

### La fonction d'arrêt des moteurs

- Cette fonction stoppe les 2 servomoteurs à l'aide de la fonction `detach()`

```
//--- fonctions de mouvements synchrones des 2 moteurs

void enAvant(int vitesseIn) { // met les 2 moteurs en avant
// vitesse entre 10 et 100%

    servoAvant(Droit, vitesseIn); // moteur Droit
    servoAvant(Gauche, vitesseIn); // moteur Gauche

} // fin en avant

void enArriere(int vitesseIn) { // met les 2 moteurs en arriere
// vitesse entre 10 et 100%

    servoArriere(Droit, vitesseIn); // moteur Droit
    servoArriere(Gauche, vitesseIn); // moteur Gauche

} // fin en arriere

void tourneDroite(int vitesseIn) { // tourne à droite
// vitesse entre 10 et 100%

    servoArriere(Droit, vitesseIn); // moteur Droit
    servoAvant(Gauche, vitesseIn); // moteur Gauche

} // fin en tourne droite

void tourneGauche(int vitesseIn) { // tourne à gauche
// vitesse entre 10 et 100%

    servoAvant(Droit, vitesseIn); // moteur Droit
    servoArriere(Gauche, vitesseIn); // moteur Gauche

} // fin tourne gauche

void servosStop() { // arret des 2 moteurs
    servo[Droit].detach(); // met le servomoteur à l'arret
    servo[Gauche].detach(); // met le servomoteur à l'arret
}
```

## Fonctions gérant fonctions reçues sur le port Série

- Cette partie du code est un peu plus longue, mais ne comporte aucun élément compliqué.
- Pour chaque fonction, on reçoit le tableau de valeurs long (les paramètres) et une valeur int représentant le nombre des paramètres
- Ensuite, on accède aux valeurs des paramètres sous la forme vars[index] tel que l'index=0 correspond au premier élément.
- On appelle, dans chaque cas, les autres fonctions utiles du programme.
- Des messages sont affichés pour améliorer le suivi de l'exécution du programme.
- Noter l'utilisation de la fonction `constrain()` qui permet d'obliger la valeur reçue en paramètre à rester entre les valeurs 0 et 100%.

**Remarquer la facilité avec laquelle on peut déclarer des fonctions qui seront attachées aux chaînes reçues sur le port Série, avec plusieurs paramètres numériques au besoin.**

```
//===== code des fonctions de controle recues sur le port Série =====

//---- fonction codeStop
void codeStop(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre

    if (qt==0) { // si aucun paramètre reçu
        Serial.println("Arret moteurs");
        servosStop();
    }
    else Serial.println("Fonction non valide ! ");
} // --- fin codeStop

//---- fonction codeAvantD
void codeAvantD(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre

    if (qt==0) { // si aucun paramètre reçu
        Serial.println("Moteur D en Avant a la vitesse 100 ");
        servoAvant(Droit, 100);
    }
    else if (qt==1) { // si 1 paramètre reçu
        Serial.print("Moteur D en Avant a la vitesse : ");
        Serial.println(constrain(vars[0],0,100)); // 1er paramètre - oblige valeur 0-100
        servoAvant(Droit,constrain(vars[0],0,100));
    }
    else Serial.println("Fonction non valide ! ");
} // --- fin codeAvantD

//---- fonction codeAvantG
void codeAvantG(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre

    if (qt==0) { // si aucun paramètre reçu
```

[Purchased by Franck Ourion, franck.ourion@univ-lorraine.fr #6280170](mailto:franck.ourion@univ-lorraine.fr)

Atelier Arduino : Apprendre à envoyer des fonctions avec paramètres en provenance du Terminal Série avec la carte Arduino.

```

    Serial.println("Moteur G en Avant a la vitesse 100 ");
    servoAvant(Gauche, 100);
}
else if (qt==1) { // si 1 paramètre reçu
    Serial.print("Moteur G en Avant a la vitesse : ");
    Serial.println(constrain(vars[0],0,100)); // 1er paramètre - oblige valeur 0-100
    servoAvant(Gauche,constrain(vars[0],0,100));
}
else Serial.println("Fonction non valide ! ");

} // --- fin codeAvantG

//---- fonction codeArriereD
void codeArriereD(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre

    if (qt==0) { // si aucun paramètre reçu
        Serial.println("Moteur D en Arriere a la vitesse 100 ");
        servoArriere(Droit, 100);
    }
    else if (qt==1) { // si 1 paramètre reçu
        Serial.print("Moteur D en Arriere a la vitesse : ");
        Serial.println(constrain(vars[0],0,100)); // 1er paramètre - oblige valeur 0-100
        servoArriere(Droit,constrain(vars[0],0,100));
    }
    else Serial.println("Fonction non valide ! ");

}

} // --- fin codeArriereD

//---- fonction codeArriereG
void codeArriereG(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre

    if (qt==0) { // si aucun paramètre reçu
        Serial.println("Moteur G en Arriere a la vitesse 100 ");
        servoArriere(Gauche, 100);
    }
    else if (qt==1) { // si 1 paramètre reçu
        Serial.print("Moteur G en Arriere a la vitesse : ");
        Serial.println(constrain(vars[0],0,100)); // 1er paramètre - oblige valeur 0-100
        servoArriere(Gauche,constrain(vars[0],0,100));
    }
    else Serial.println("Fonction non valide ! ");

}

} // --- fin codeArriereG

//---- fonction codeEnAvant
void codeEnAvant(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre

    if (qt==0) { // si aucun paramètre reçu
        Serial.println("Moteur D et G en Avant a la vitesse 100 ");

```

```

    enAvant(100);
}
else if (qt==1) { // si 1 paramètre reçu
    Serial.print("Moteur D et G en Avant a la vitesse : ");
    Serial.println(constrain(vars[0],0,100)); // 1er paramètre - oblige valeur 0-100
    enAvant(constrain(vars[0],0,100));
}
else Serial.println("Fonction non valide ! ");

} // --- fin codeEnAvant

//---- fonction codeEnArriere
void codeEnArriere(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre

    if (qt==0) { // si aucun paramètre reçu
        Serial.println("Moteur D et G en Arriere a la vitesse 100 ");
        enArriere(100);
    }
    else if (qt==1) { // si 1 paramètre reçu
        Serial.print("Moteur D et G en Arriere a la vitesse : ");
        Serial.println(constrain(vars[0],0,100)); // 1er paramètre - oblige valeur 0-100
        enArriere(constrain(vars[0],0,100));
    }
    else Serial.println("Fonction non valide ! ");

} // --- fin codeEnArriere

//---- fonction codeTourneD
void codeTourneD (long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre

    if (qt==0) { // si aucun paramètre reçu
        Serial.println("Tourne vers la Droite a la vitesse 100 ");
        tourneDroite(100);
    }
    else if (qt==1) { // si 1 paramètre reçu
        Serial.print("Tourne vers la Droite a la vitesse : ");
        Serial.println(constrain(vars[0],0,100)); // 1er paramètre - oblige valeur 0-100
        tourneDroite(constrain(vars[0],0,100));
    }
    else Serial.println("Fonction non valide ! ");

} // --- fin codeTourneD

//---- fonction codeTourneG
void codeTourneG(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre

    if (qt==0) { // si aucun paramètre reçu
        Serial.println("Tourne vers la Gauche a la vitesse 100 ");
        tourneGauche(100);
    }
}

```

```

else if (qt==1) { // si 1 paramètre reçu
  Serial.print("Tourne vers la Gauche a la vitesse : ");
  Serial.println(constrain(vars[0],0,100)); // 1er paramètre - oblige valeur 0-100
  tourneGauche(constrain(vars[0],0,100));
}
else Serial.println("Fonction non valide ! ");

} // --- fin codeTourneG

//---- fonction testServoD
void testServoD(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre

  if (qt==1) { // si 1 paramètre reçu
    Serial.print("Largeur impulsion servo Droit = ");
    Serial.print(vars[0]);
    Serial.println(" microsecondes.");

    servo[Droit].attach(brocheServo[Droit]); // attache le servomoteur à la broche
    servo[Droit].writeMicroseconds(vars[0]);
  }
  else Serial.println("Fonction non valide ! ");
} // --- fin testServoD

//---- fonction testServoG
void testServoG(long *vars, int qt) { // vars est le tableau de paramètres et qt leur nombre

  if (qt==1) { // si 1 paramètre reçu
    Serial.print("Largeur impulsion servo Gauche = ");
    Serial.print(vars[0]);
    Serial.println(" microsecondes.");

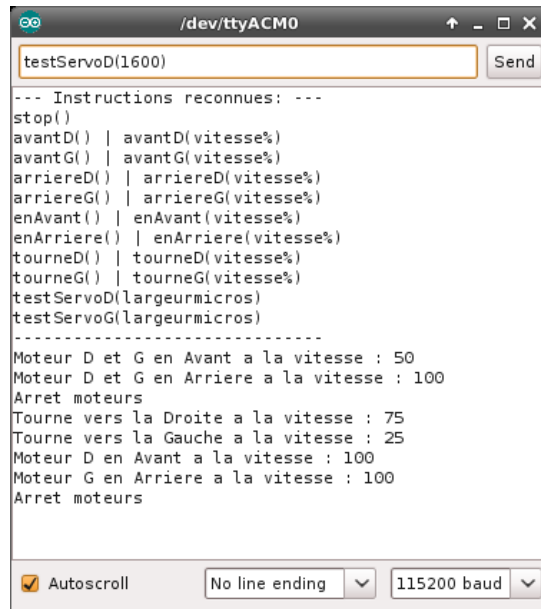
    servo[Gauche].attach(brocheServo[Gauche]); // attache le servomoteur à la broche
    servo[Gauche].writeMicroseconds(vars[0]);
  }
  else Serial.println("Fonction non valide ! ");
} // --- fin testServoG

```



## Fonctionnement du programme

- Ouvrir le Terminal Série (Tools > Serial Monitor) et fixer le débit à la même valeur que celle utilisée pour l'instruction `Serial.begin(vitesse)`. Ici, **115200** bauds.
- Régler également les paramètres de transmission de la chaîne de caractère à l'aide de la 2ème liste défilante. **Mettre sur « New Line »** pour ajout du « saut de ligne » après la chaîne saisie.
- saisir l'une des chaînes reconnues en mettant entre parenthèses les valeurs souhaitées : les moteurs réalisent l'action demandée ! Très très pratique pour tester son petit robot par le port Série de manière très fine.
- Noter la possibilité d'étalonner les servomoteurs au besoin avec ce programme avec les instructions `testServoD()` et `testServoG()`

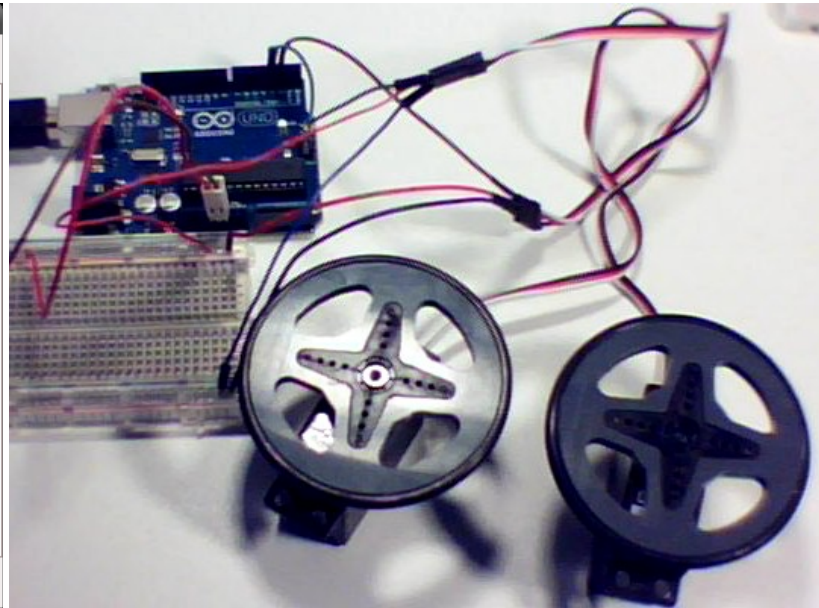


```

/dev/ttyACM0
testServoD(1600) Send

--- Instructions reconnues: ---
stop()
avantD() | avantD(vitesse%)
avantG() | avantG(vitesse%)
arriereD() | arriereD(vitesse%)
arriereG() | arriereG(vitesse%)
enAvant() | enAvant(vitesse%)
enArriere() | enArriere(vitesse%)
tourneD() | tourneD(vitesse%)
tourneG() | tourneG(vitesse%)
testServoD(largeurmicros)
testServoG(largeurmicros)
-----
Moteur D et G en Avant a la vitesse : 50
Moteur D et G en Arriere a la vitesse : 100
Arret moteurs
Tourne vers la Droite a la vitesse : 75
Tourne vers la Gauche a la vitesse : 25
Moteur D en Avant a la vitesse : 100
Moteur G en Arriere a la vitesse : 100
Arret moteurs

Autoscroll No line ending 115200 baud
```



Les chaînes de caractères contrôlent les rotations individuelles ou synchrones des 2 servomoteurs à rotation continue !  
Les valeurs numériques reçues en paramètres fixent la vitesse de rotation de 0 à 100% !

Les bases du contrôle de dispositifs via le port Série sont ici posées.

Il sera dès lors possible de réaliser le contrôle depuis une interface graphique sur le PC (Processing).

Il sera également possible d'envoyer des chaînes par communication sans fil (XBee), par réseau ethernet ou wifi, en utilisant des cartes d'extensions adaptées.

## 18. Les éléments du langage Arduino étudiés dans cet atelier

Structure	Variables et constantes	Fonctions
<b>Structures de contrôle</b> <ul style="list-style-type: none"><li>• <a href="#">while</a></li><li>• <a href="#">do... while</a></li><li>• <a href="#">break</a></li></ul>	<b>Type des données</b> <ul style="list-style-type: none"><li>• <a href="#">char</a></li><li>• <a href="#">objet String</a></li></ul>	<b>Librairie <a href="#">Serial</a></b> <ul style="list-style-type: none"><li>• <a href="#">begin()</a></li><li>• <a href="#">available()</a></li><li>• <a href="#">read()</a></li><li>• <a href="#">flush()</a></li><li>• <a href="#">print()</a></li><li>• <a href="#">println()</a></li></ul>

La documentation complète du langage Arduino en français est disponible ici :  
[http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n=Main.ReferenceMaxi](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.ReferenceMaxi)

## **19. *A présent, vous devriez être capable :***

- d'écrire un programme capable de recevoir des fonctions avec paramètres multiples en provenance du Terminal Série avec Arduino

# Table des matières

Intro |  
Matériel nécessaire pour les ateliers Arduino |  
Rappel : Principe de communication de l'Arduino vers le PC |  
Rappel : Notion de « Classe » |  
Rappel : la classe Serial |  
Rappel : « Hello world ! » : programme Arduino envoyant un message vers le PC via le port USB |  
Rappel : Lancer et paramétrer le Terminal Série pour afficher/recevoir des messages entre le PC et Arduino |  
Rappel : Langage Arduino : Introduction aux bibliothèques |  
Découvrir et installer les bibliothèques Arduino fournies par la communauté Arduino |  
Exemple d'installation et présentation d'une bibliothèque de la communauté : la bibliothèque RunF |  
Réception d'une fonction avec paramètres numériques sur le port Série : le montage |  
Réception d'une fonction avec 1 paramètre numérique sur le port Série : le programme |  
Réception d'une fonction avec plusieurs paramètres numériques entiers sur le port Série : le programme |  
Contrôler une LED multicolore RGB par le port série : le montage |  
Contrôler une LED multicolore RGB par le port série : le programme |  
Contrôle de la vitesse et du sens de 2 servomoteurs à rotation continue par le port Série : le montage |  
Contrôle de la vitesse et du sens de 2 servomoteurs à rotation continue par le port Série : le programme |  
Les éléments du langage Arduino étudiés dans cet atelier |  
A présent, vous devriez être capable : |

**Bravo !**  
vous avez terminé cet atelier Arduino !



Prêt pour la suite ? Retrouvez de nombreux autres thèmes d'ateliers Arduino ici :

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.ATELIERS](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS)