



roboonly

# Communication Serial

## Filaire

## Par Bluetooth

# Pourquoi communiquer ?

- Problèmes:
  - comment **connaître la valeur d'une variable** au cours de l'exécution d'un programme (debug) ?
  - comment **envoyer des informations** à un ordinateur ou un robot ?
  - comment **en recevoir** ?
- Il nous faut un moyen/protocole de communication !

**On utilise alors la communication série**

# C'est quoi ?

- Communication série ou **Serial**
- Utilise un *port série* ou *port COM* présent sur tous les ordinateurs (physique, via USB, ou virtuel)
- Dans notre cas, l'information est transmise caractère par caractère en suivant le format ASCII.



## Pour les uC, comment ça marche ?

- L'information est transmise **octet par octet**  
(rappel: 1 octet = 8 bits)
- Chaque octet, en arrivant chez le destinataire, est stockée dans un **buffer**  
(ou **mémoire tampon**)
- Le destinataire lit ensuite les octets présents dans le buffer par ordre d'arrivée



roboonly

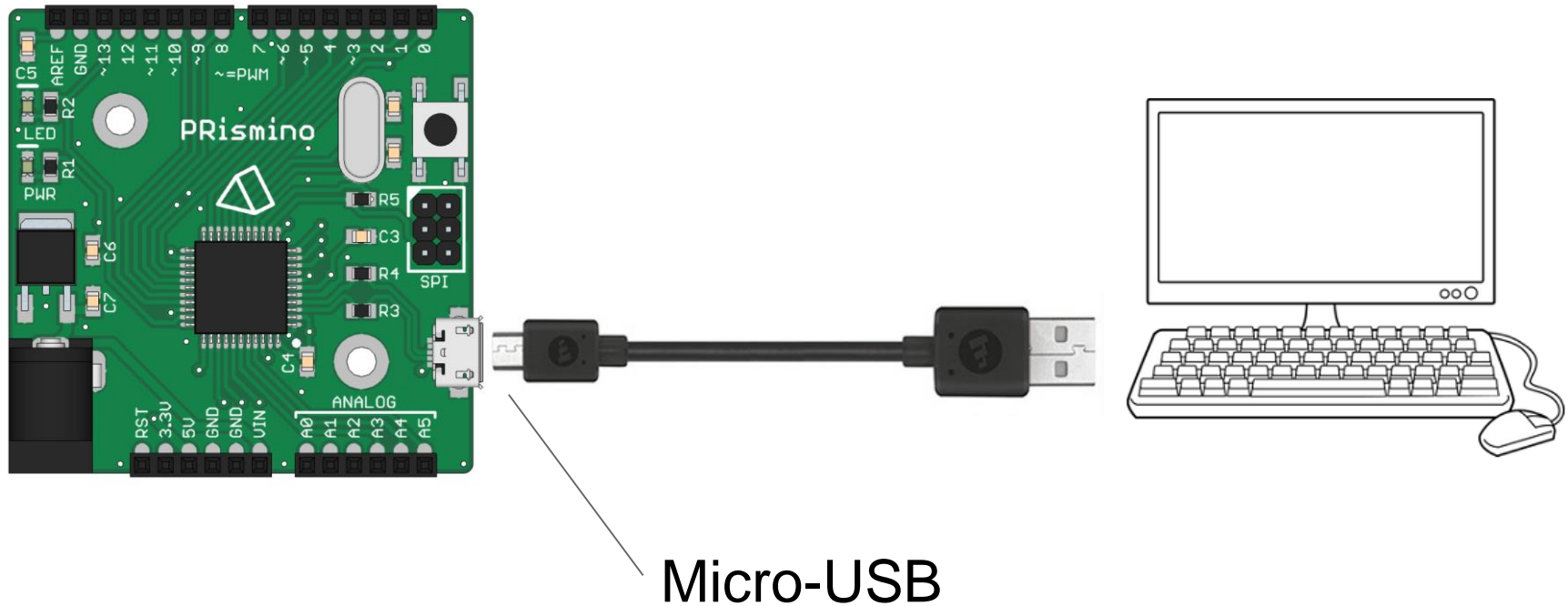
# Communication Serial

## Filaire

## Par Bluetooth

# Hardware

- Tout simplement par USB



# Envoi du microcontrôleur

- Plusieurs fonctions de la bibliothèque Arduino peuvent être appelées
- Rappel de la **structure de base** d'un programme

```
void setup()
```

```
{
```



Fonction exécutée qu'une seule fois, au lancement du programme

```
}
```

```
void loop()
```

```
{
```



Fonction exécutée en boucle : c'est le code principal

```
}
```

# Envoi du microcontrôleur

- Initialisation de la communication Serial
  - `Serial.begin (vitesse) ;`
  - `vitesse` = vitesse de communication  
= généralement `9600`
  - appeler une fois par programme dans `setup`
- Envoyer de l'information
  - `Serial.print (X) ;`
  - `X` peut être une variable, un caractère, une chaîne...



# Envoi du microcontrôleur

- **Exemples** d'envois :
  - `Serial.print (ma_variable) ;`
  - `Serial.print ("texte") ;`
  - `Serial.print (456) ;`
  - `Serial.print ('a') ;`
- **Variante** : `Serial.println (X) ;`
  - **Saute une ligne** après l'information **x**
  - Équivalent à ajouter le caractère `\n`

# Envoi du microcontrôleur

- Et pour les **nombre à virgules** ?
  - Par défaut **deux décimales** sont envoyées
  - Mais on peut choisir :
    - `Serial.print(x, nb_décimales);`
    - `nb_décimales` = nombre de décimales de `x` à envoyer

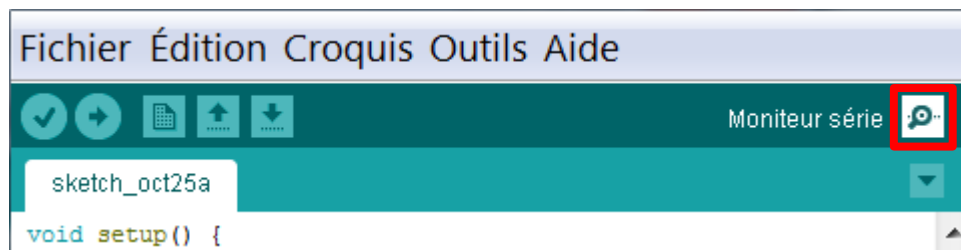
```
Serial.print(1.23456, 0); //prints "1"
```

```
Serial.print(1.23456, 2); //prints "1.23"
```

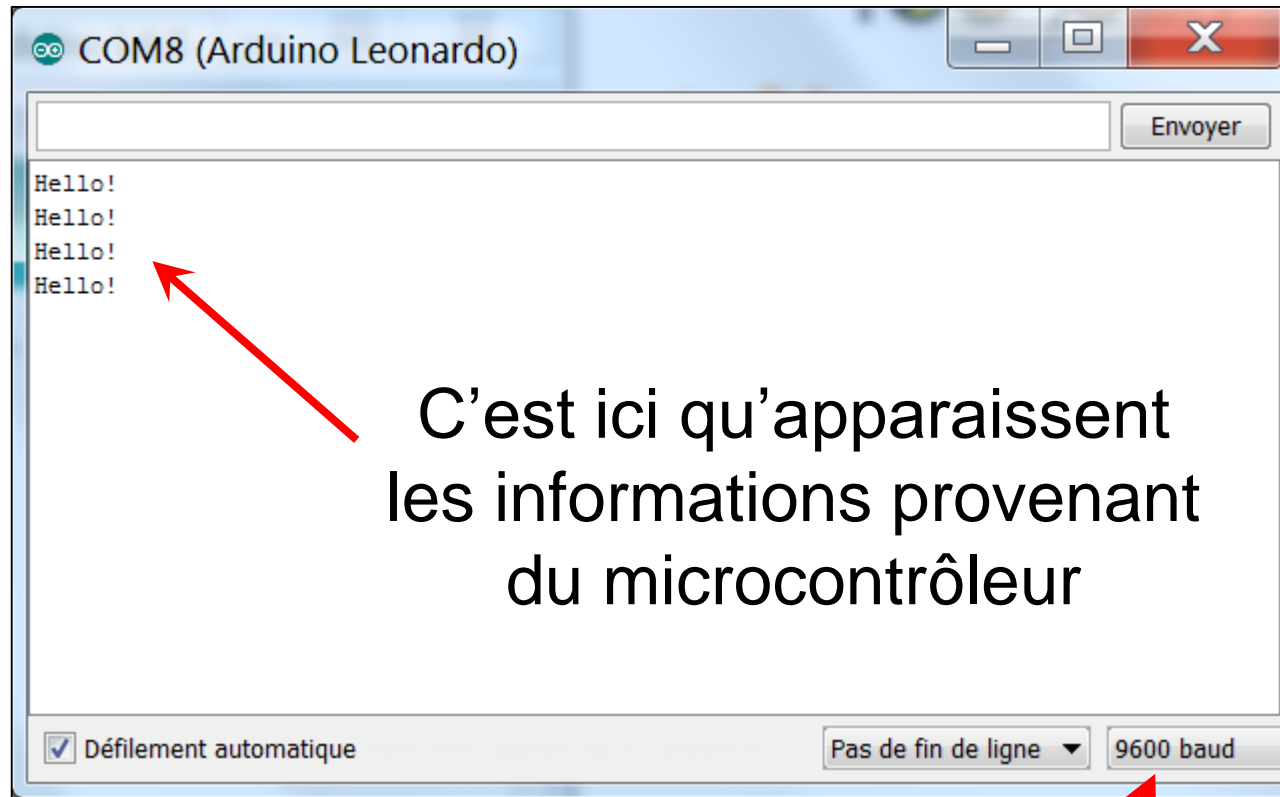
```
Serial.print(1.23456, 4); //prints "1.2346"
```

# Envoi du microcontrôleur

- Lire les données **depuis un ordinateur** :
  - Ouvrir **Arduino IDE**
  - Vérifier que le microcontrôleur est bien reconnu (connecté à un Port COM)
  - Ouvrir le **Moniteur série**



# Envoi du microcontrôleur



Choix de la vitesse de communication

# Envoi du microcontrôleur - exemples

- Maintenant, quelques exemples !
- Mais tout d'abord des rappels :
  - `pinMode (numero , mode) ;`  
 → `mode = INPUT` ou `OUTPUT`
  - `digitalRead (numero) ;`
  - `analogRead (numero) ;`  
 → renvoie une valeur entre 0 et 1023
  - `digitalWrite (numero , état) ;`  
 → `état = HIGH` ou `LOW`

# Lire la valeur du potentiomètre

```
#include <prismo.h>

void setup()
{
    pinMode(POT, INPUT);
    Serial.begin(9600);
}

void loop()
{
    Serial.print("valeur = ");
    Serial.println(analogRead(POT));
    delay(400);
}
```



**Inclure la  
bibliothèque  
prismo.h**

# Lire la valeur du potentiomètre

```


#include <prismo.h>

void setup()
{
  pinMode(POT, INPUT);
  Serial.begin(9600);
}

void loop()
{
  Serial.print("valeur = ");
  Serial.println(analogRead(POT));
  delay(400);
}

```

**Définir le mode  
du pin utilisé  
(POT en INPUT)**



# Lire la valeur du potentiomètre

```
#include <prismo.h>

void setup()
{
  pinMode(POT, INPUT);
  Serial.begin(9600);
}

void loop()
{
  Serial.print("valeur = ");
  Serial.println(analogRead(POT));
  delay(400);
}
```

**Définir la vitesse  
de communication  
(ici 9600)**





# Lire la valeur du potentiomètre

```
#include <prismo.h>

void setup()
{
  pinMode(POT, INPUT);
  Serial.begin(9600);
}

void loop()
{
  Serial.print("valeur = ");
  Serial.println(analogRead(POT));
  delay(400);
}
```

**Envoyer**  
une chaîne  
de caractère



# Lire la valeur du potentiomètre

```
#include <prismo.h>

void setup()
{
  pinMode(POT, INPUT);
  Serial.begin(9600);
}

void loop()
{
  Serial.print("valeur = ");
  Serial.println(analogRead(POT));
  delay(400);
}
```

Envoyer  
la valeur  
du POT




# Lire la valeur du potentiomètre

```
#include <prismo.h>

void setup()
{
    pinMode(POT, INPUT);
    Serial.begin(9600);
}

void loop()
{
    Serial.print("valeur = ");
    Serial.println(analogRead(POT));
    delay(400);
}
```



Attendre 400 millisecondes

# Lire la valeur du potentiomètre

```

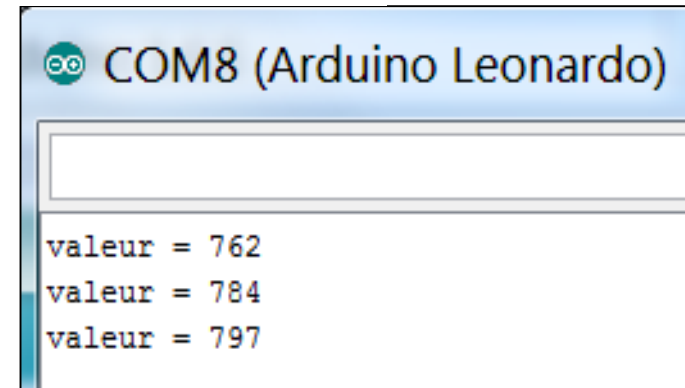
#include <prismo.h>

void setup()
{
    pinMode(POT, INPUT);
    Serial.begin(9600);
}

void loop()
{
    Serial.print("valeur = ");
    Serial.println(analogRead(POT));
    delay(400);
}

```

## Résultat :



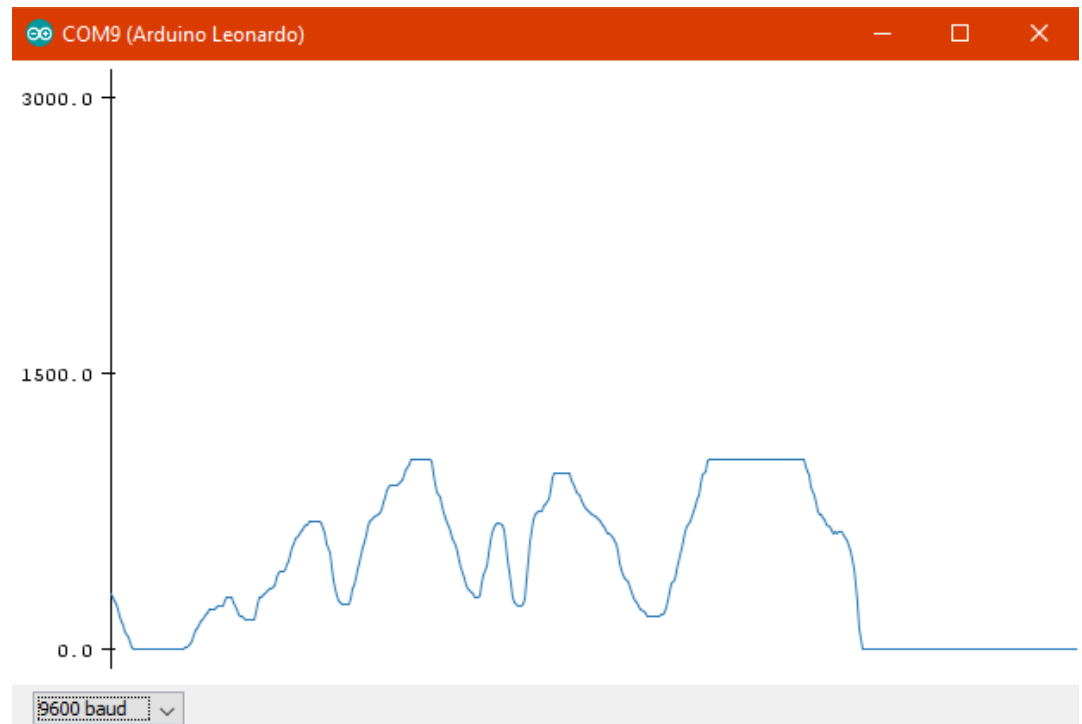
# Lire la valeur du potentiomètre

Meilleur moyen pour lire des données continues sur l'IDE d'Arduino depuis la version 1.6.6

Le Serial Plotter (*Traceur Série* en Français)

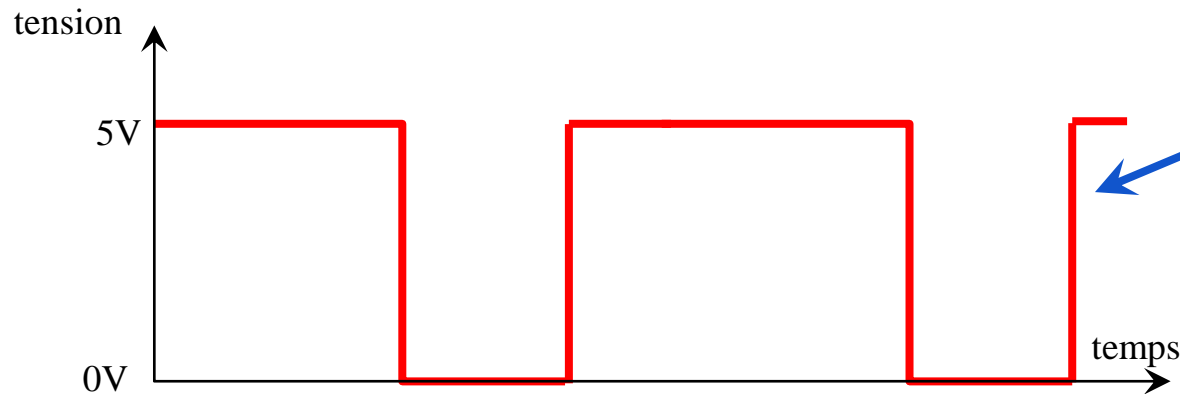
**Exemple** avec le potentiomètre

**Ne pas afficher des strings !**



# Compter un évènement

- But : **compter** le nombre de fois que le bouton est pressé
- Que se passe-t-il sur le pin lorsque l'évènement se produit ?



Bouton relâché  
= de 0 (0V) à 1 (5V)  
*flanc montant*

Bouton pressé = de 1 (5V) à 0 (0V)  
appelé *flanc descendant*

**Il faut arriver à distinguer les deux cas!**


# Compter un évènement

```
int count = 0;
bool old_value = true, new_value = true;

void setup() {
  pinMode(BTN, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  new_value = digitalRead(BTN);
  if(new_value != old_value && !new_value) {
    count++;
    Serial.print("count = ");
    Serial.println(count);
  }
  old_value = new_value;
}
```

**Définir le mode du pin utilisé**  
(attention au cas particulier du **BTN!**)



# Compter un évènement

```
int count = 0;
bool old_value = true, new_value = true;

void setup() {
  pinMode(BTN, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  new_value = digitalRead(BTN);
  if(new_value != old_value && !new_value) {
    count++;
    Serial.print("count = ");
    Serial.println(count);
  }
  old_value = new_value;
}
```

**count**: compteur  
**new\_value**: valeur du bouton au début de chaque itération  
**old\_value**: valeur précédente



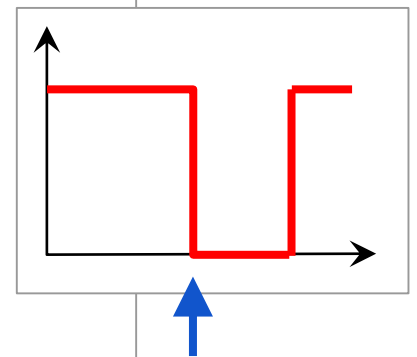
# Compter un évènement

```
int count = 0;
bool old_value = true, new_value = true;

void setup() {
  pinMode(BTN, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  new_value = digitalRead(BTN);
  if(new_value != old_value && !new_value) {
    count++;
    Serial.print("count = ");
    Serial.println(count);
  }
  old_value = new_value;
}
```

On doit savoir s'il y a eu **changement d'état du bouton** et détecter s'il a été **pressé ou relaché**



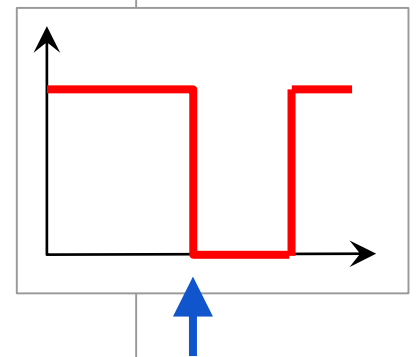
# Compter un évènement

```
int count = 0;
bool old_value = true, new_value = true;

void setup() {
  pinMode(BTN, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  new_value = digitalRead(BTN);
  if(new_value != old_value && !new_value) {
    count++;
    Serial.print("count = ");
    Serial.println(count);
  }
  old_value = new_value;
}
```

Si c'est le cas, on **incrémente le compteur** et on **envoie la nouvelle valeur**



# Compter un évènement

```

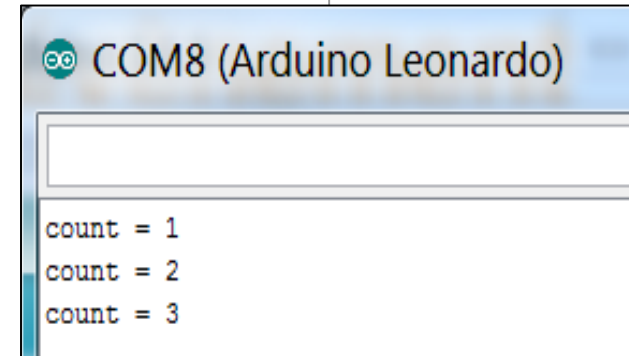
int count = 0;
bool old_value = true, new_value = true;

void setup() {
  pinMode(BTN, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  new_value = digitalRead(BTN);
  if(new_value!=old_value && !new_value) {
    count++;
    Serial.print("count = ");
    Serial.println(count);
  }
  old_value = new_value;
}

```

**Résultat :**



COM8 (Arduino Leonardo)

```

count = 1
count = 2
count = 3

```

# Réception du microcontrôleur

- Pour recevoir, c'est plus compliqué. Il faut :
  - Vérifier si des octets sont arrivés dans le buffer
  - Les lire et les retirer du buffer
- **Serial.available** ()
  - renvoie le nombre d'octets en attente dans le buffer
- **Serial.read** ()
  - renvoie le premier octet du buffer et le retire

# Réception du microcontrôleur

- Pour lire un caractère on procède donc ainsi:


```
char lettre;  
...  
if (Serial.available ())  
{  
    lettre =  
    Serial.read ();  
}  
...
```

Si le buffer n'est **pas vide**,  
la valeur renvoyée sera  
**différente de 0**, donc  
condition **true**

# Réception du microcontrôleur

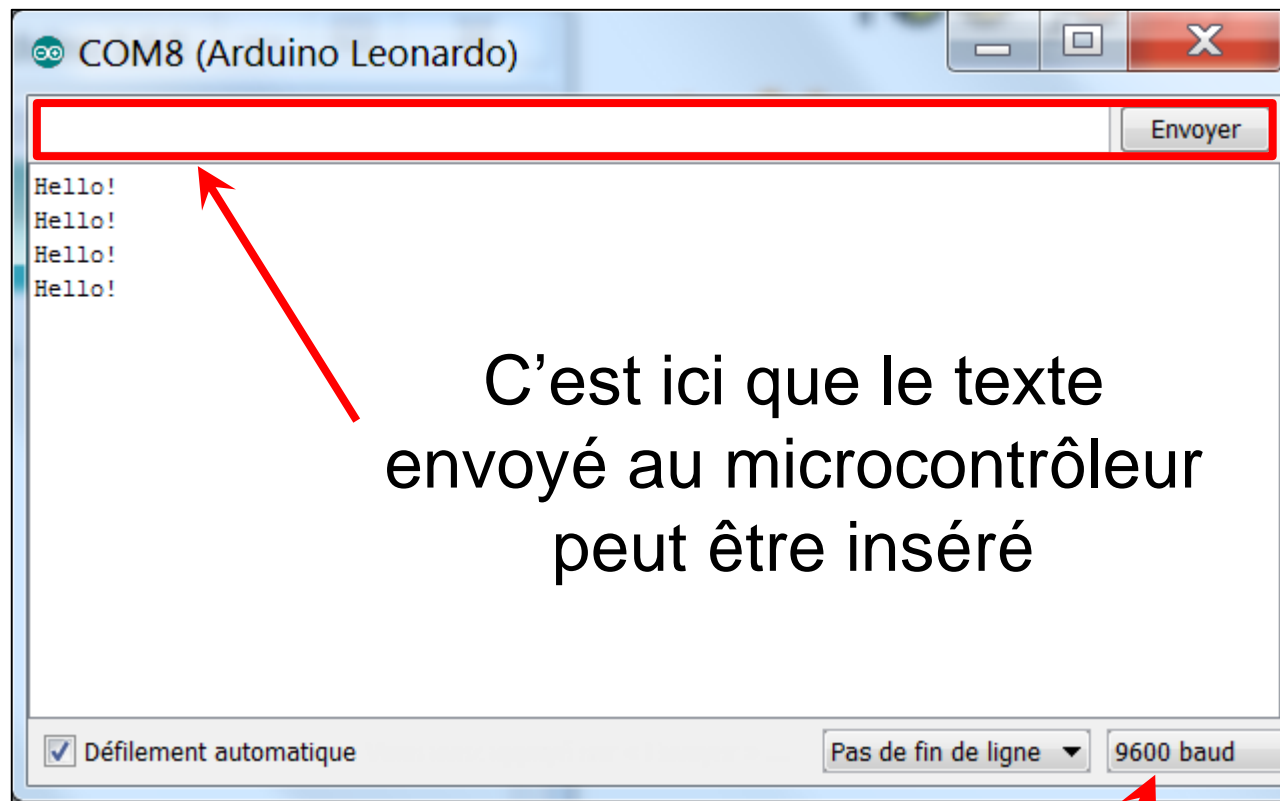
- Pour lire un caractère on procède donc ainsi:

```
char lettre;  
...  
if (Serial.available())  
{  
    lettre =  
    Serial.read();  
}  
...
```



**Lire** le caractère  
du buffer

# Réception du microcontrôleur



Choix de la vitesse de communication

# Allumer une LED

```
#include <prismo.h>
char c;
void setup() {
    pinMode(LED, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    if(Serial.available()) {
        c = Serial.read();
        if(c == '0')
            digitalWrite(LED,
LOW);
        else
            digitalWrite(LED,
HIGH);
    }
}
```

Un caractère est  
**disponible dans  
le buffer!**





# Allumer une LED

```
#include <prismo.h>
char c;
void setup() {
    pinMode(LED, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    if(Serial.available()) {
        c = Serial.read();
        if(c == '0')
            digitalWrite(LED,
LOW);
        else
            digitalWrite(LED,
HIGH);
    }
}
```

Lecture du  
caractère




# Allumer une LED

```
#include <prismo.h>
char c;
void setup() {
    pinMode(LED, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    if(Serial.available()) {
        c = Serial.read();
        if(c == '0')
            digitalWrite(LED, LOW);
            delay(1000);
        else
            digitalWrite(LED,
HIGH);
    }
}
```

**Modifier l'état**  
de la LED en  
fonction de la  
valeur reçue



# Réception du microcontrôleur

- Attention : l'ordinateur envoie seulement l'information sous forme de caractères ASCII
- Par exemple, 1 correspond au nombre 49
- On ne peut donc **pas faire**:

```
int nombre;  
...  
if (Serial.available ())  
{  
    nombre =  
    Serial.read ();  
}  
...
```

Si l'ordinateur **envoie 1**, cela sera interprété comme le **caractère '1'**, correspondant dans la table ASCII au **nombre 49**

# Réception du microcontrôleur

- Comment **recevoir un nombre** dont on ne connaît pas la taille à l'avance ?
  - On **convertit les octets reçus** dans le buffer en une chaîne de caractères
  - On **interprète** cette chaîne de caractère **comme un nombre**

# Réception du microcontrôleur

- `Serial.readBytesUntil` (`carac`, `tableau`, `n`) ;
  - Lit les octets présents dans le buffer jusqu'au caractère `carac` et les stocke dans le `tableau`
  - Si `carac` n'est pas trouvé après avoir stocké `n` caractères, la fonction ne lit pas plus loin le buffer
  - Si `tableau` est de taille `N`, `n = N-1` (attention au caractère de fin de chaîne de caractère `'\0'`)
- `atoi` (`tableau`) ;

# Réception du microcontrôleur

- Pour lire un nombre on procède donc ainsi:

```
char tab[16];
int nombre;
...
if (Serial.available ())
{
    Serial.readBytesUntil ( '\n' , tab , 15 ) ;
    nombre = atoi ( tab ) ;
}
...
```



roboonly

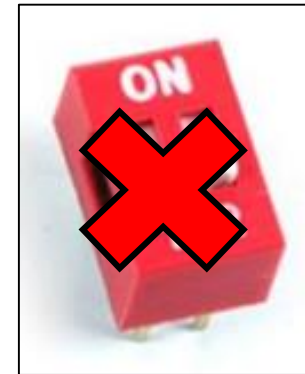
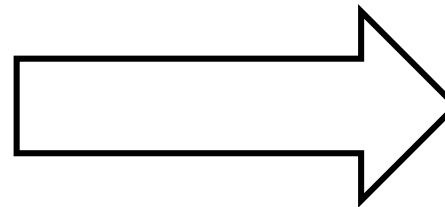
Communication Serial

Filaire

Par Bluetooth

# Module Bluetooth

- Peut être soudé derrière le shield
- Ne peut pas être soudé en même temps que le Switch



- Communication avec un ordinateur ou un autre robot



# Module Bluetooth - Généralités

- Instructions de montage, bibliothèque et documentation :

[robopoly.epfl.ch/prisme/tutoriels/bluetooth](http://robopoly.epfl.ch/prisme/tutoriels/bluetooth)

- Le module a un **nom** et un **mot de passe** propres
- L'ordinateur détecte le module et peut **s'y connecter** en donnant le mot de passe correct

# Module Bluetooth - Configuration

- Utiliser le fichier exemple ***BluetoothConfig*** de la bibliothèque
- Permet de configurer **par Serial USB** le module lors de sa première utilisation
- Depuis le *Moniteur Série*, modifier le **nom**, **mot de passe** et **vitesse de communication**

# Module Bluetooth - Utilisation

**Bonne nouvelle** : côté software, ça  
fonctionne exactement **comme le**  
**Serial USB !**

# Module Bluetooth - Utilisation

- Cependant, **petite différence** :
  - dans chaque fonction vue précédemment, remplacer **Serial.** par **Bluetooth.**
  - *exemple:* remplacer **Serial.read()** par **Bluetooth.read()**
- Exception:
  - **Bluetooth.print(x)** n'existe pas
  - il faut utiliser **Bluetooth.write(x)** où **x** ne peut être qu'une chaîne de caractère

Chaîne de destination

Taille max chaîne

Format

Nombre à traduire

- → nécessité de convertir les nombres en chaînes avec **snprintf(tableau, size, "%d", nombre);**

# Module Bluetooth - Utilisation

- Depuis **Arduino IDE** :
  - Choisir le **Port COM Bluetooth** auquel le module est connecté
  - Ouvrir le ***Moniteur série***
  - Communiquer !
- Depuis  **votre propre programme** :
  - Tout à fait possible dans n'importe quel langage avec une **bibliothèque gérant la communication Serial**
  - **QtSerialPort** marche bien, depuis le framework Qt (gratuit pour un usage non commercial)

# À vos projets !

**Pas d'idée ? Visitez notre page projet !**

[robopoly.epfl.ch/projets](http://robopoly.epfl.ch/projets)

**Trop d'idées ? Venez nous en faire part !**

[projects@robopoly.ch](mailto:projects@robopoly.ch)

# Prochains événements

## ● Workshop II

- Samedi 11 novembre, 9h-18h
- En haut du BM !
- Un lien doodle vous sera envoyé
- Pour poursuivre le montage de ton robot, discuter de tes projets, demander de l'aide au comité !

## ● Demon

- Lundi prochain, 12h15, ELA 1
- Utiliser les machines du local + concevoir une base mécanique

# Contact/Infos

**Contact principal**

[robopoly@epfl.ch](mailto:robopoly@epfl.ch)

**Site officiel - toutes les infos et slides sont la!**

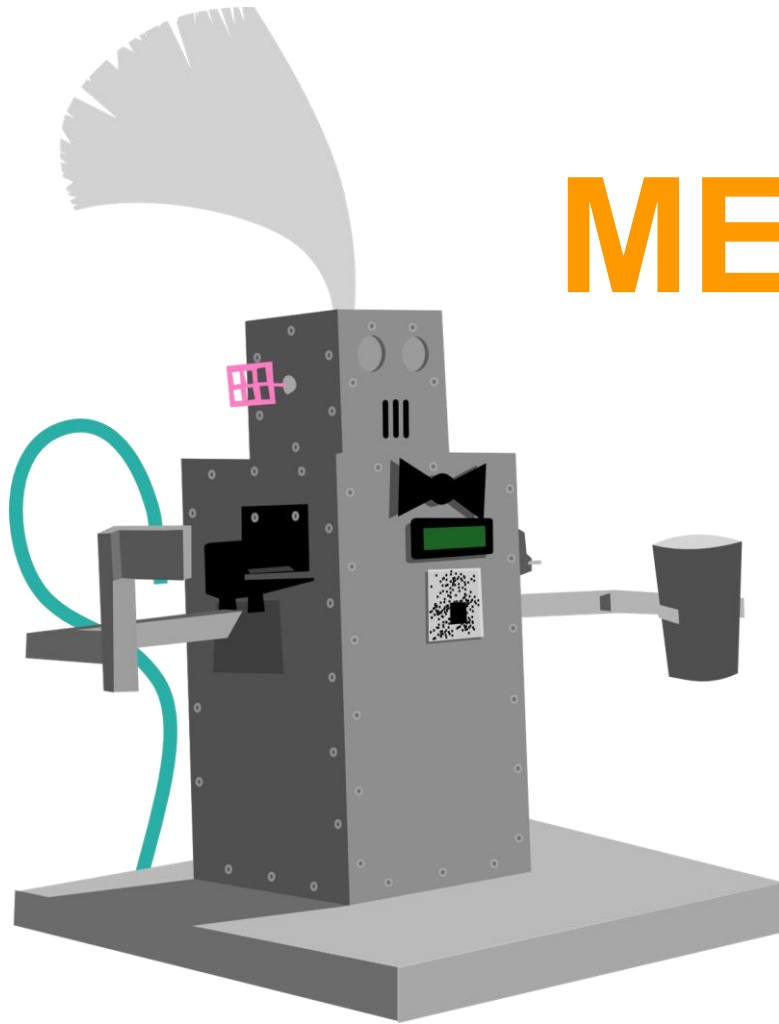
[robopoly.epfl.ch](http://robopoly.epfl.ch)

**Facebook - pour suivre l'actualité du club!**

[www.facebook.com/robopoly](http://www.facebook.com/robopoly)



MERCI!



Questions ?