

Apprendre à **recevoir des nombres** en provenance du Terminal Série avec la carte Arduino, apprendre à **contrôler Arduino** à partir du Terminal Série, apprendre à utiliser **l'interface graphique Processing** pour interagir graphiquement avec Arduino.



Ateliers Arduino

par X. HINAULT

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'avez pas payé 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

Pour obtenir tout autres types de licence d'utilisation (enseignement, commercial, etc...), veuillez contacter l'auteur ici : 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

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 comprendre le principe de réception de valeurs numériques entières en provenance du Terminal Série
- d'apprendre à contrôler Arduino depuis le port Série par une chaîne de caractère ou une valeur numérique
- découvrir le principe de contrôle de la carte Arduino depuis le port Série à partir d'une interface graphique Processing

... afin d'être en mesure d'interagir avec la carte Arduino à partir du PC et d'être en mesure ensuite de contrôler facilement des dispositifs (moteurs, etc...)

Cet atelier fait logiquement suite à l'atelier « Recevoir des chaînes de caractères en provenance du Terminal Série avec Arduino »

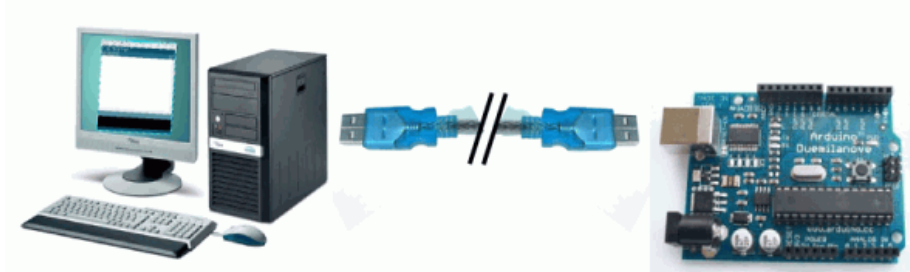


Prêt ? C'est parti !

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

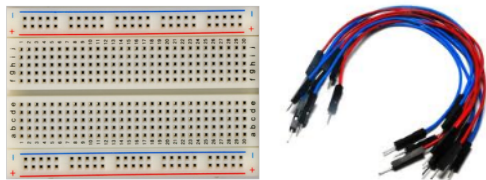


L'espace de développement Arduino associe :

- 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

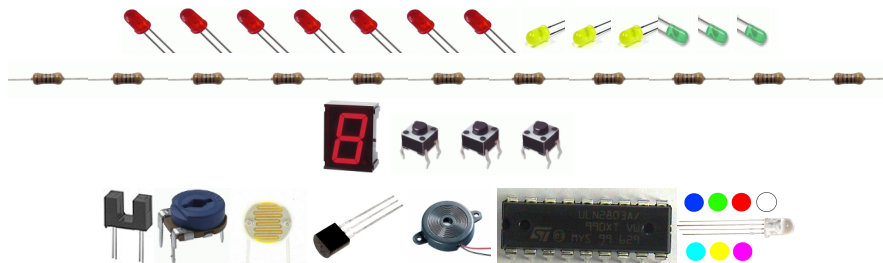


Pour réaliser des montages sans soudure, vous aurez besoin :

- 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

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) !

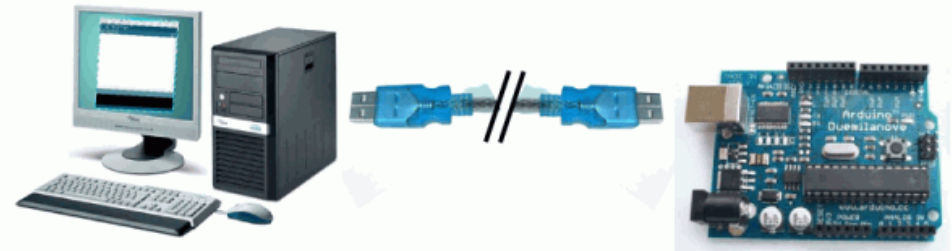
Le langage Arduino dispose de toutes les instructions nécessaires pour réaliser et programmer cette communication au sein d'un programme comme nous allons le voir ici.

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.

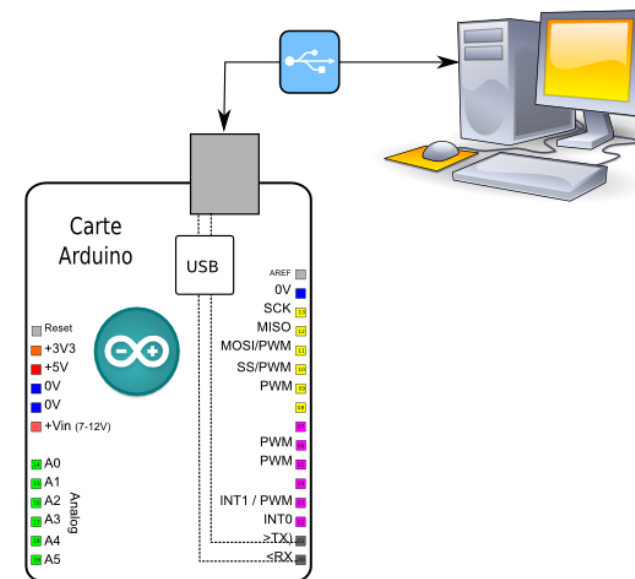
Au final, la carte Arduino programmée pourra fonctionner de 2 façons :

- 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.

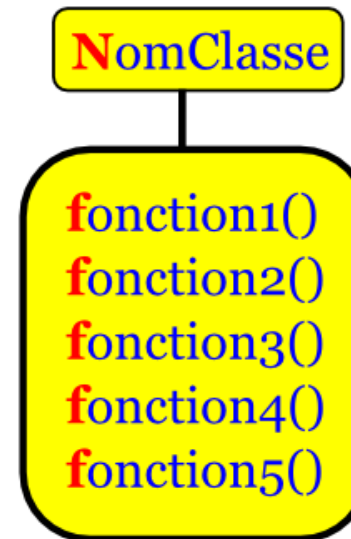
Tout comme une fonction, une classe aura un nom : pour distinguer une classe d'une fonction, **le nom d'une classe commencera par une MAJUSCULE.**

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 (rappel)

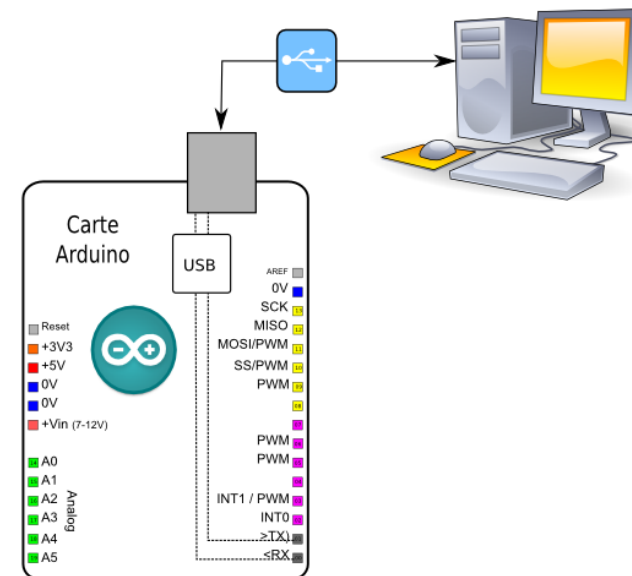
Ici, nous allons découvrir les fonctions de la classe Serial utiles en réception :

- `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 nombres en provenance du PC depuis notre carte Arduino !



6. La fonction **Serial.available()**

Description

Donne le nombre d'octets (caractères) disponibles pour lecture dans la file d'attente (buffer) du port série. (available veut dire disponible en anglais)

Syntaxe

Serial.available();

Paramètres

Aucun (laisser les parenthèses vides)

Valeur renvoyée

Renvoie nombre d'octet disponible pour lecture dans la file d'attente (buffer) du port série, ou 0 si aucun caractère n'est disponible. Si une donnée est arrivée, Serial.available() sera supérieur à 0. La file d'attente du buffer peut recevoir jusqu'à 128 octets.

Type : int

Exemple

```
if (Serial.available() > 0) { // si des données entrantes sont présentes
    // lit le 1er octet arrivé
    incomingByte = Serial.read();
    // ici instructions à exécuter
}
```

Voir également :

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



7. La fonction **Serial.read()**

Description

Lit les données entrantes sur le port Série.

Syntaxe

Serial.read() ;

Paramètres

Aucun

Valeur renvoyée

Renvoie le premier octet de donnée entrant disponible dans le buffer du port série, ou -1 si aucune donnée n'est disponible.

Type : int

L'octet lu est « enlevé » de la file d'attente. Le prochain appel de la fonction `read()` lira l'octet suivant, etc...

Exemple

```
if (Serial.available() > 0) { // si des données entrantes sont présentes
    // lit le 1er octet arrivé
    incomingByte = Serial.read();
    // ici instructions à exécuter
}
```

Voir également :

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



8. Rappel : Pour info : Le code ASCII (American Standard Code for Information Interchange)

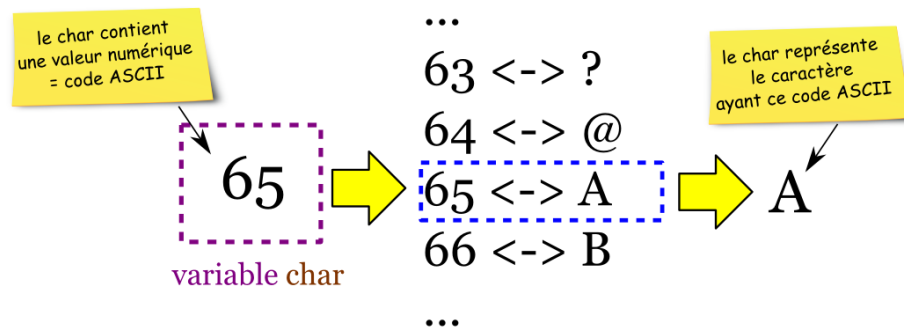
Le code ASCII

Historiquement, pour représenter les caractères sur un système informatique, on a utilisé un codage simplifié où à 1 nombre correspondait un caractère. L'intérêt de cette façon de faire, c'est de permettre de coder les caractères sur 1 seul octet (au lieu de coder tous les pixels du caractère...).

Dec	Hx	Oct	Html	Chr
64	40	100	@	@
65	41	101	A	A
66	42	102	B	B
67	43	103	C	C
68	44	104	D	D
69	45	105	E	E
70	46	106	F	F

Correspondance entre le code ASCII, le type char et le caractère

ASCII <-> caractère



Pour faire simple, on peut dire qu'une variable de type char contient un caractère, et c'est ce que vous devez retenir, tout en sachant que la variable char contient en fait le code ASCII du caractère !

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
40	28	050	((72	48	110	H	H	104	68	150	h	h
41	29	051))	73	49	111	I	I	105	69	151	i	i
42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z

Les caractères spéciaux

Certains caractères dits « spéciaux » sont à connaître, notamment :

- le saut de ligne (New Line) – code ASCII = 10
- le retour de chariot (Carriage Return) – code ASCII = 13

Un truc très utile : pour les caractères 0 à 9, on obtient la valeur numérique correspondante en faisant (code ASCII – 48)

9. Rappel : Réglage et Utilisation du Terminal Série en « émission » vers Arduino

Une fois que la carte sera programmée, on va configurer le Terminal Série pour fonctionner en « émission » vers Arduino.

Lancement et réglage du Terminal Série en Réception

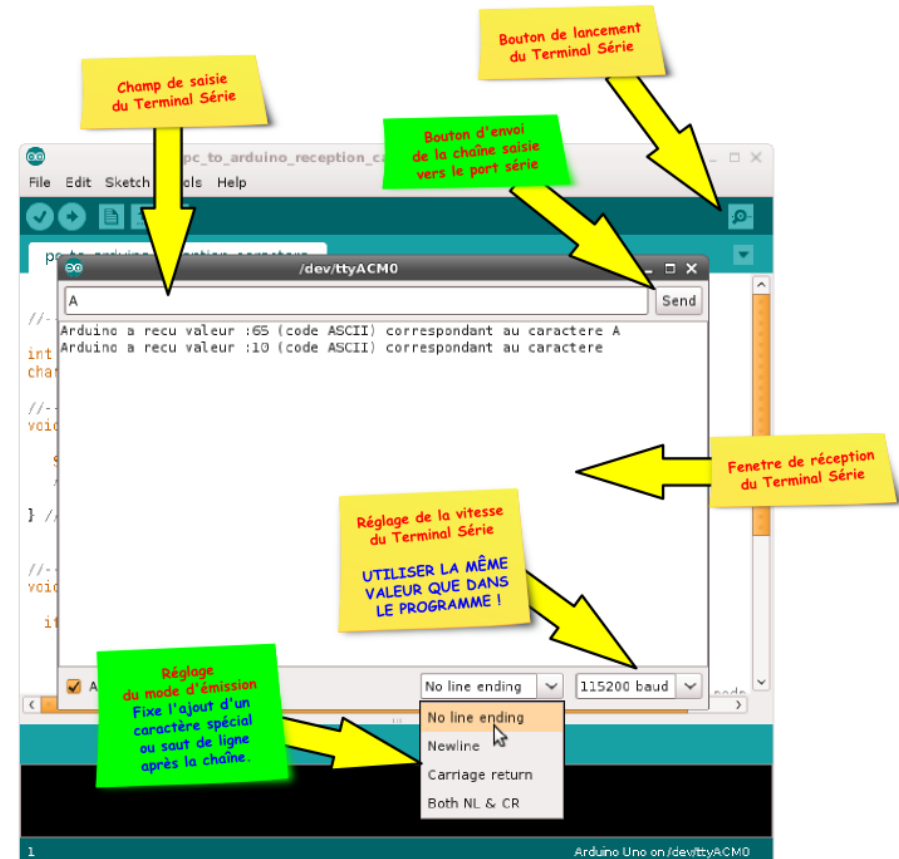
- Ouvrir le Terminal Série (Tools > Serial Monitor) et fixer le débit à la même valeur que celle utilisée pour l'instruction **Serial.begin**(vitesse). Ici, 115200 bauds.
- Régler également les paramètres de transmission de la chaîne de caractère à l'aide de la 2ème liste déroulante :
 - sur « **No Line Ending** » afin qu'aucun caractère de fin de ligne ne soit émis après la chaîne de caractères,
 - sur « **New Line** » si on souhaite qu'un saut de ligne soit émis après la chaîne de caractères (envoi le caractère ascii=10 après la chaîne)
 - sur « **Carriage Return** » si on souhaite qu'un retour de chariot soit émis après la chaîne de caractères (envoi le caractère ascii=13 après la chaîne)
 - sur « **Both NL & CR** » pour les 2 simultanément.

Envoi d'une chaîne sur le port Série

- Ensuite se positionner dans le champ de saisie et saisir 1 ou plusieurs caractères dans le champ
- Clic sur envoi : la chaîne est émise sur le port Série +/- suivie des options de fin de ligne.

Visualisation de la réponse d'Arduino

- Si Arduino renvoie une réponse, celle-ci s'affiche dans la fenêtre de visualisation du Terminal



10. Programme : Recevoir une valeur numérique entière sur le port Série

Recevoir une chaîne de caractères est une chose... recevoir une valeur numérique par le port série en est une autre. Réfléchissez bien : si vous saisissez « 12345 » et que vous l'envoyez sur le port série... vous envoyez en fait les caractères '1' puis '2' puis '3' puis '4' puis '5' et c'est ce qu'Arduino va recevoir... Comment alors récupérer la valeur numérique correspondante ??

Une idée ?? Allez, je vous aide :

Un truc très utile : pour les caractères 0 à 9, on obtient la valeur numérique correspondante en faisant (code ASCII – 48)

Entete déclarative

- on déclare une variable **int** pour stocker l'octet en réception (code ASCII du caractère), une variable **char** pour stocker le caractère correspondant et une variable **long** pour stocker le nombre en réception.

Fonction **setup()**

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

Fonction **loop()**

- A ce niveau, on va « écouter » le port Série en testant l'arrivée d'un caractère à l'aide d'une boucle **while** pour tester la présence d'un octet dans la file d'attente du port série avec la fonction **Serial.available()**
- Tant qu'un octet différent du saut de ligne est présent on ajoute la caractère à l'objet String. On réalise une petite pause entre 2 réceptions.
- Si c'est un saut de ligne que l'on reçoit, on sort de la boucle **while**.
- Une fois toute la chaîne reçue, on l'affiche.

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.

```
//--- entete déclarative = variables et constantes globales
int octetReception=0; // variable de réception octet
char caractereReception=0; // variable de réception caractère
long nombreReception=0; // déclare variable long stocker nombre reçu

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

    Serial.begin(115200); // initialise la vitesse de la connexion série
    //-- utilise la meme vitesse dans le Terminal Série

} // fin de la fonction setup()

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

    while (Serial.available()>0) { // si un caractère en réception

        octetReception=Serial.read(); // lit le 1er octet de la file d'attente

        if (octetReception==10) { // si Octet reçu est le saut de ligne
            Serial.print ("Saut de ligne reçu : ");
            Serial.print ("Nombre reçu = "); // affiche la chaîne reçue
            Serial.println (nombreReception);
            nombreReception=0; //RAZ le String de réception
            delay(100); // pause
            break; // sort de la boucle while
        } // fin if

        else { // si le caractère reçu n'est pas un saut de ligne

            octetReception=octetReception-48; // transfo valeur ASCII en
            valeur décimale

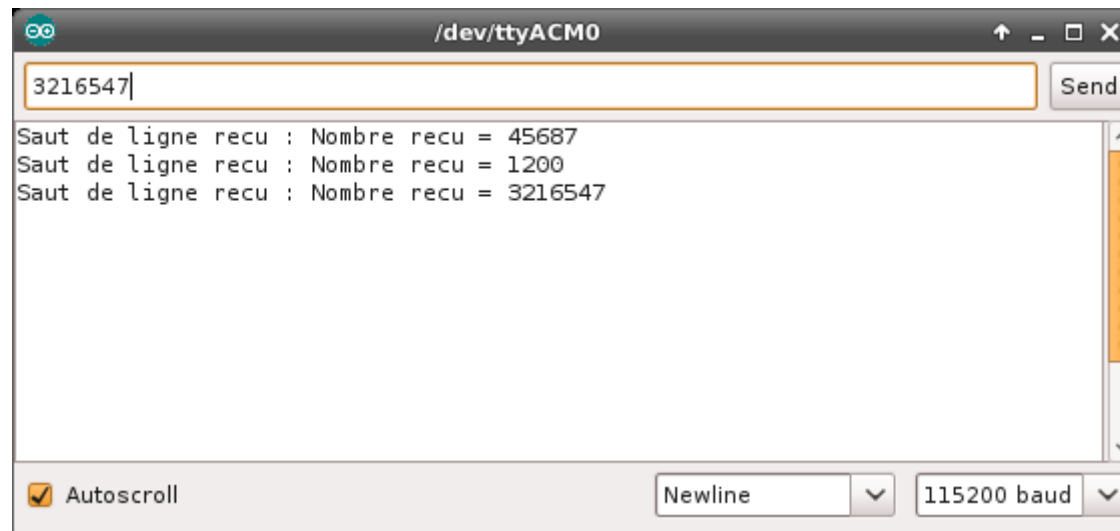
            // calcul du nombre à partir des valeurs reçues
            if ((octetReception>=0)&&(octetReception<=9)) nombreReception =
            (nombreReception*10)+octetReception;
            else Serial.println("La chaîne n'est pas un nombre valide !");

            delay(1); // laisse le temps au caractères d'arriver

        } // fin else

    } // fin while - fin de réception de la chaîne

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



Je vous laisse dès lors imaginer toutes les possibilités d'interactions possibles entre le PC et Arduino :

- régler la valeur d'une variable (un seuil d'alarme, un code secret, une coordonnée, une heure, etc...)
- régler la fréquence d'un son, contrôler un appareil de musique, etc...
- régler la position d'un moteur, sa vitesse, etc...
- contrôler les mouvements d'un robot, etc...

**De plus, à partir d'une simple interface Processing,
il sera possible de faire la même chose mais à partir de la souris ou de boutons de réglage, etc..**

Combien ça coûte déjà une carte Arduino ? 20€ ???? quoi ! tout ça pour 20€... ?!
chut, faut pas dire... ça reste entre nous...



11. Utile : Une fonction « clé en main » pour lire une valeur numérique + signe sur le port Série

Bon allez, je vais faire le gentil : voici une fonction « clé en main » que je vous propose pour recevoir une valeur numérique entière signée sur le port Série. Gardez là sous le coude pour l'intégrer dans vos programmes en cas de besoin...

```
long recevoirNombre() { // fonction de reception d'un nombre sur le port série

    int octetRecu=0; // variable pour octet reçu
    int compt=0; // variable locale comptage caractères reçus
    boolean signe=true; // variable locale signe nombre reçu
    long nombreRecu=0; // variable locale nombre reçu

    while (Serial.available()>0) { // tant qu'un octet en réception

        octetRecu=Serial.read(); // Lit le 1er octet reçu et le met dans la variable

        if (octetReception==10) { // si Octet reçu est le saut de ligne

            break; // sort de la boucle while
        }
        else { // si le caractère reçu n'est pas un saut de ligne

            if ((octetRecu=='-') && (compt==0))signe=false; // si Octet reçu est le - et si c'est le 1er caractère reçu - signe négatif
            compt=compt+1; // incrémente compt

            octetRecu=octetRecu-48; // transfo valeur ASCII en valeur décimale

            // calcul du nombre à partir des valeurs reçues
            if ((octetRecu>=0)&&(octetRecu<=9)) nombreRecu = (nombreRecu*10)+octetRecu;

        }

        delay(1); // pause pour laisser le temps à la fonction available de recevoir octet suivant
    } // fin tant que octet réception

    // après le break déclenché par saut de ligne, on se retrouve ici

    if (signe==false) nombreRecu=nombreRecu*(-1); // prise en compte signe négatif

    return(nombreRecu); // renvoie le nombre calculé
} // fin fonction recevoirNombre
```

12. Contrôler une LED à partir du Terminal Série : le montage

Ce qu'on va faire...

Pour mettre en évidence le principe d'interaction entre Arduino et le PC, ou comment contrôler Arduino à partir du PC, je vous propose quelque chose de simple.

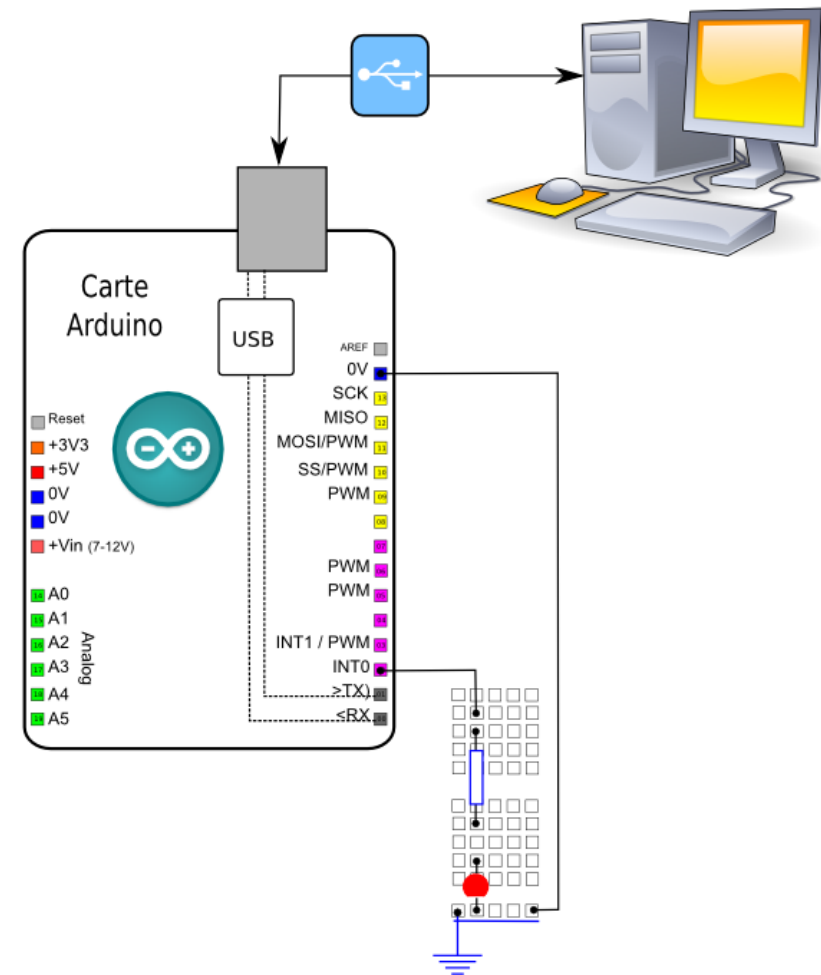
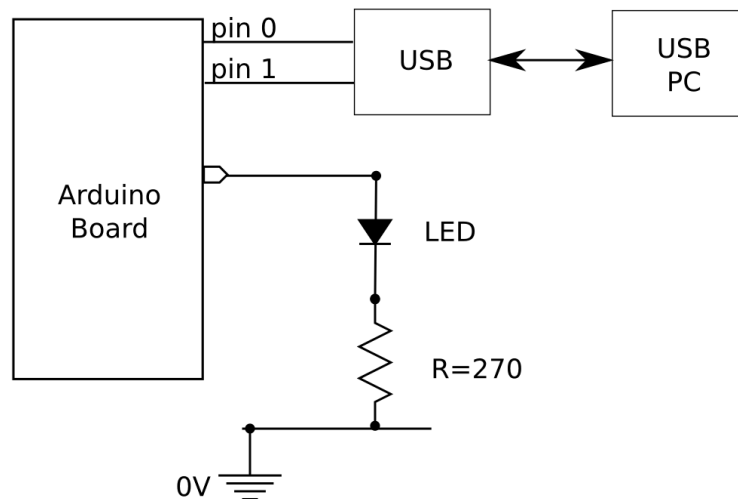
On va envoyer une chaîne depuis le Terminal série vers Arduino qui devra allumer ou éteindre une LED selon la chaîne reçue :

- la chaîne « LED=ON » allumera la LED
- la chaîne « LED=OFF » éteindra la LED

Tout bête, mais ça vous donne les bases pour des choses beaucoup plus évoluées !

Schéma théorique du montage

Une LED est connectée sur une broche numérique en sortie et la carte Arduino communique avec le PC sur le port Série :



13. Contrôler une LED à partir du Terminal Série : le programme

Reprenons le code de réception d'une chaîne avec saut de ligne déjà vue ([Voir l'atelier « Recevoir des chaînes de caractères en provenance du Terminal Série avec Arduino »](#)). Il va suffire de tester le contenu de la chaîne pour contrôler la LED. Allons-y !

Entête déclarative

- On déclare :
 - une variable **int** pour stocker l'octet en réception (code ASCII du caractère),
 - une variable **char** pour stocker le caractère correspondant,
 - un objet **String** vide pour stocker la chaîne de caractère
- On déclare également une constante de broche pour la LED

```
//--- entete déclarative = variables et constantes globales

int octetReception=0; // variable de réception octet
char caractereReception=0; // variable de réception caractère
String chaineReception=""; // déclare un objet String vide

const int LED=2; // constante de broche pour LED
```

Fonction **setup()**

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

```
//--- 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

    pinMode(LED, OUTPUT); // met la LED en sortie

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


Fonction `loop()`

- A ce niveau, on va « écouter » le port Série en testant l'arrivée d'un caractère à l'aide d'une boucle `while` pour tester la présence d'un octet dans la file d'attente du port série avec la fonction `Serial.available()`
- Tant qu'un octet différent du saut de ligne est présent on ajoute la caractère à l'objet String. **On réalise une petite pause entre 2 réceptions.**
- Et si c'est un saut de ligne que l'on reçoit :
 - on va tester la chaîne reçue et exécuter l'action voulue
 - on l'affiche. puis on sort de la boucle `while`.

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

void loop() {

    while (Serial.available()>0) { // si un caractère en réception

        octetReception=Serial.read(); // lit le 1er octet de la file
d'attente

        if (octetReception==10) { // si Octet reçu est le saut de ligne
            Serial.print ("Saut de ligne reçu : ");
            Serial.println ("Chaîne recue = "+chaîneReception); // affiche
la chaîne recue

            //--- allume/éteint la LED si la chaîne attendue est reçue
            if (chaîneReception=="LED=ON") {
                digitalWrite(LED, HIGH);
                Serial.println (" Allume LED !");
            }
            if (chaîneReception=="LED=OFF") {
                digitalWrite(LED, LOW);
                Serial.println (" Eteint LED !");
            }

            chaîneReception=""; //RAZ le String de réception
            delay(100); // pause
            break; // sort de la boucle while
        } // fin if

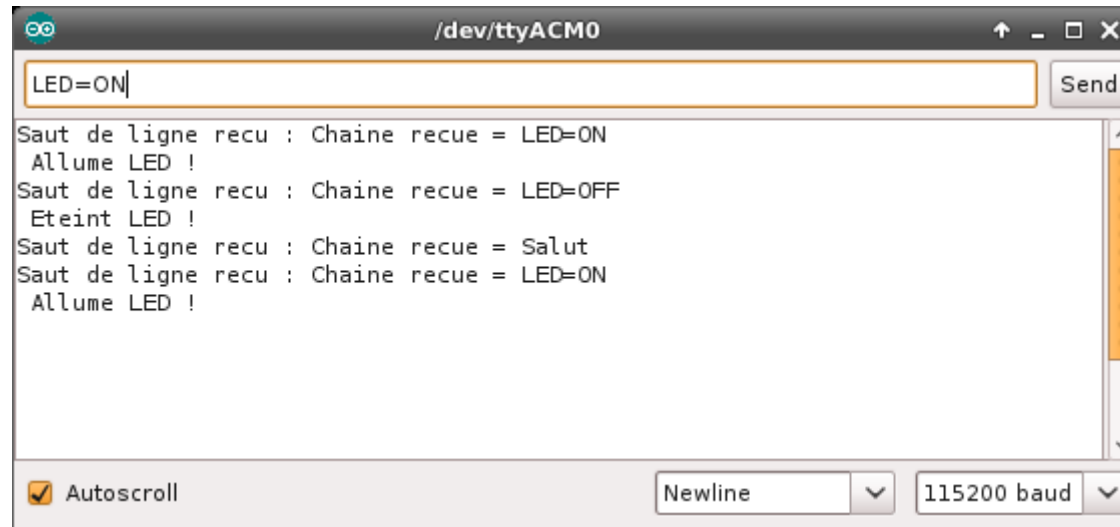
        else { // si le caractère reçu n'est pas un saut de ligne
            caractereReception=char(octetReception); // récupere le caractere
à partir du code Ascii
            chaîneReception=chaîneReception+caractereReception; // ajoute la
caractère au String
            delay(1); // laisse le temps au caractères d'arriver
        } // fin else

    } // fin while - fin de réception de la chaîne

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

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.
- si on saisit LED=ON, la LED s'allume, LED=OFF l'éteint !



LED=ON : La LED s'allume
LED=OFF : La LED s'éteint
Toute autre chaîne est ignorée !

14. Fixer la fréquence de clignotement de la LED à partir du Terminal Série : le programme.

A présent, faisons quelque chose de comparable mais en transmettant à Arduino une valeur numérique par le port Série. Le montage utilisé sera donc le même que précédemment. La valeur numérique que nous allons transmettre fixera la vitesse de clignotement de la LED. Reprenons le programme de réception d'une valeur numérique sur le port Série.

Entête déclarative

- On déclare :
 - une variable **int** pour stocker l'octet en réception (code ASCII du caractère),
 - une variable **char** pour stocker le caractère correspondant,
 - une variable **long** pour stocker le nombre reçu
 - une variable **int** pour stocker la vitesse de clignotement (en ms)
- On déclare également une constante de broche pour la LED

```
//--- entete déclarative = variables et constantes globales
int octetReception=0; // variable de réception octet
char caractereReception=0; // variable de réception caractère
String chaineReception=""; // déclare un objet String vide
long nombreReception=0; // déclare variable long stocker nombre reçu

int vitesse=500; // vitesse de clignotement de la LED

const int LED=2; // broche de la LED
```

Fonction **setup()**

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

```
//--- 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

    pinMode(LED, OUTPUT); // met la LED en sortie

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

Fonction `loop()`

Gestion du port Série

- A ce niveau, on va « écouter » le port Série en testant l'arrivée d'un caractère à l'aide d'une boucle `while` pour tester la présence d'un octet dans la file d'attente du port série avec la fonction `Serial.available()`
- Tant qu'un octet différent du saut de ligne est présent on ajoute la valeur du nombre à la valeur courante x 10. **On réalise une petite pause entre 2 réceptions.**
- Et si c'est un saut de ligne que l'on reçoit :
 - on affiche le nombre reçu
 - on met à jour la variable vitesse, on l'affiche.
 - puis on sort de la boucle `while`.

Clignotement de la LED

- une fois le port Série tester, on gère le clignotement de la LED, la vitesse de clignotement étant fixée par la variable vitesse.

Noter que les caractères reçus sur le port Série sont mémorisés en « file d'attente » tant que le port Série n'est pas lu : aucun caractère ne sera donc ignoré, même si il arrive en dehors du moment où l'on teste le port Série.

En conséquence, il y aura un petit décalé entre le moment où l'on saisi la valeur et le moment où elle est prise en compte, notamment pour les valeurs élevées (pause de clignotement longue).

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

    while (Serial.available()>0) { // si un caractère en réception

        octetReception=Serial.read(); // lit le 1er octet de la file
        d'attente

        if (octetReception==10) { // si Octet reçu est le saut de ligne
            Serial.print ("Saut de ligne reçu : ");
            Serial.print ("Nombre reçu = "); // affiche la le nombre reçu
            Serial.println (nombreReception);

            vitesse=nombreReception; // modifie la valeur de la variable
            vitesse

            Serial.print ("Vitesse = "); // affiche la chaine
            Serial.println (vitesse); // affiche la variable vitesse

            nombreReception=0; //RAZ le String de réception
            delay(10); // pause
            break; // sort de la boucle while
        } // fin if

        else { // si le caractère reçu n'est pas un saut de ligne

            octetReception=octetReception-48; // transfo valeur ASCII en
            valeur décimale

            // calcul du nombre à partir des valeurs reçues
            if ((octetReception>=0)&&(octetReception<=9)) nombreReception
            = (nombreReception*10)+octetReception;
            else Serial.println("La chaine n'est pas un nombre valide !");

            delay(1); // laisse le temps au caractères d'arriver

        } // fin else

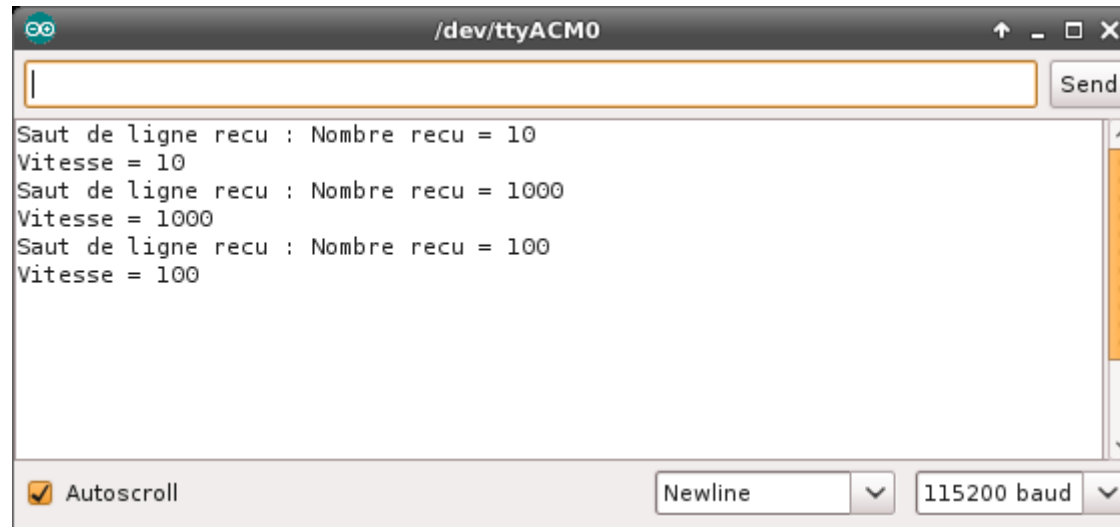
    } // fin while - fin de réception de la chaine

    //----- clignotement de la LED
    digitalWrite(LED, HIGH);
    delay(vitesse);
    digitalWrite(LED, LOW);
    delay(vitesse);

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

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 déroulante. **Mettre sur « New Line »** pour ajout du « saut de ligne » après la chaîne saisie.
- saisir une valeur numérique en millisecondes dans le champ de saisie et appui sur <send> : le clignotement de la LED est modifié en conséquence !



Ce programme peut permettre de tester la « persistance rétinienne » :
tester différentes valeurs pour voir à quelle vitesse de clignotement l'oeil ne le perçoit plus.



Noter que de la même façon, on pourrait contrôler l'intensité de la LED, la fréquence d'un son... à partir de la valeur reçue sur le port Série !

15. Introduction à Processing

Pour aller un petit peu plus loin, à titre d'exemple, je vous propose de voir comment écrire un programme dans l'interface Processing pour interagir facilement avec votre carte Arduino.

Processing, c'est quoi ?

Processing est une interface graphique programmable, libre et open source, qui va permettre par programmation de réaliser des dessins, manipuler et modifier des images, de la vidéo, de la 3D, de gérer le clavier ou la souris... et même de **communiquer avec Arduino par le port Série !!** C'est cette fonction qui est particulièrement intéressante et que nous allons utiliser ici.

Le langage de programmation s'appelle le langage Processing et ressemble dans ses principes de base au langage Arduino, même si il y a des différences évidemment.

Un programme Processing, ça fonctionne comment ?

Pour faire simple, un programme Processing en exécution va ouvrir une fenêtre graphique sur votre ordinateur et va la rafraîchir à intervalle régulier.

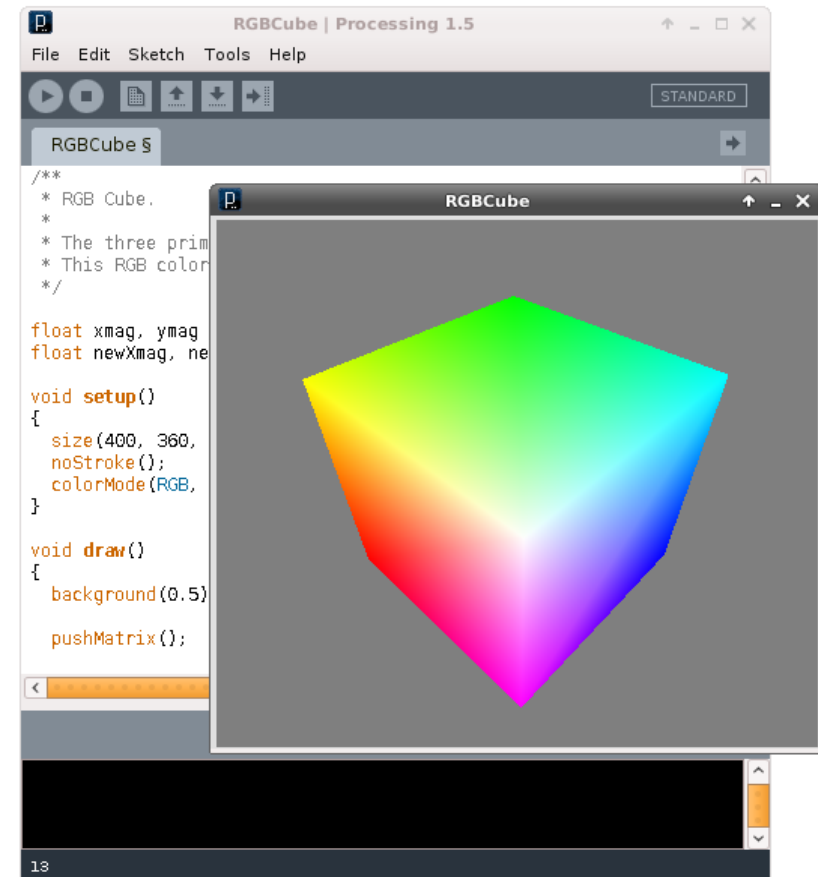
Un programme Processing va avoir la même structure qu'un programme Arduino :

- une entête déclarative
- une fonction **setup()** exécutée une seule fois au démarrage
- une fonction **draw()** qui sera exécutée à intervalle régulier (60 fois par seconde par défaut) et qui va redessiner la fenêtre principale du programme.

L'interface Processing

L'interface Processing (à télécharger sur <http://processing.org>) ressemble étrangement à l'interface Arduino : c'est normal, car l'interface Arduino est dérivée de l'interface Processing !! Vous allez donc retrouver :

- une **zone d'édition** du programme
- une **console** pour afficher des messages
- et une **barre de boutons** avec un bouton d'exécution, un bouton de stop, un bouton de création d'un fichier du programme exécutable (au lieu du bouton de transfert vers Arduino inutile ici...)



L'exemple cube RGB

(dispo dans File > Examples > 3D > Form > cube RGB)

Processing est écrit en Java et est « multi-OS » fonctionnant aussi bien sous Windows, Mac Os X que Linux (Ubuntu notamment).

16. Processing : le programme type

Le programme Processing de base ressemble étrangement à un programme Arduino... donc vous allez facilement y arriver !

Entete déclarative

- on inclura à ce niveau les bibliothèques utilisées par le programme : une bibliothèque est une « extension » du langage utilisable à la demande.
- on déclarera à ce niveau les variables et objets utilisés par le programme

Fonction **setup()**

- cette fonction est exécutée 1 seule fois au début du lancement du programme. A ce niveau on fixe les paramètres graphiques que l'on va utiliser dans le programme.
- une fonction importante que l'on mettra à ce niveau : la fonction **size**(largeur, hauteur) qui fixe la taille de la fenêtre Processing.

Fonction **draw()**

- Cette fonction est le coeur du programme et est appelée à intervalle régulier, 60 fois par seconde par défaut (réglable au besoin).
- **Le programme Processing est exécuté sur votre ordinateur, et non plus sur une carte Arduino...** La fonction **draw()**, comme son nom l'indique, va donc redessiner la fenêtre Processing à l'écran avec le code qu'elle contient : on réalisera très facilement des animations de la sorte !

Fonctions de gestion des événements

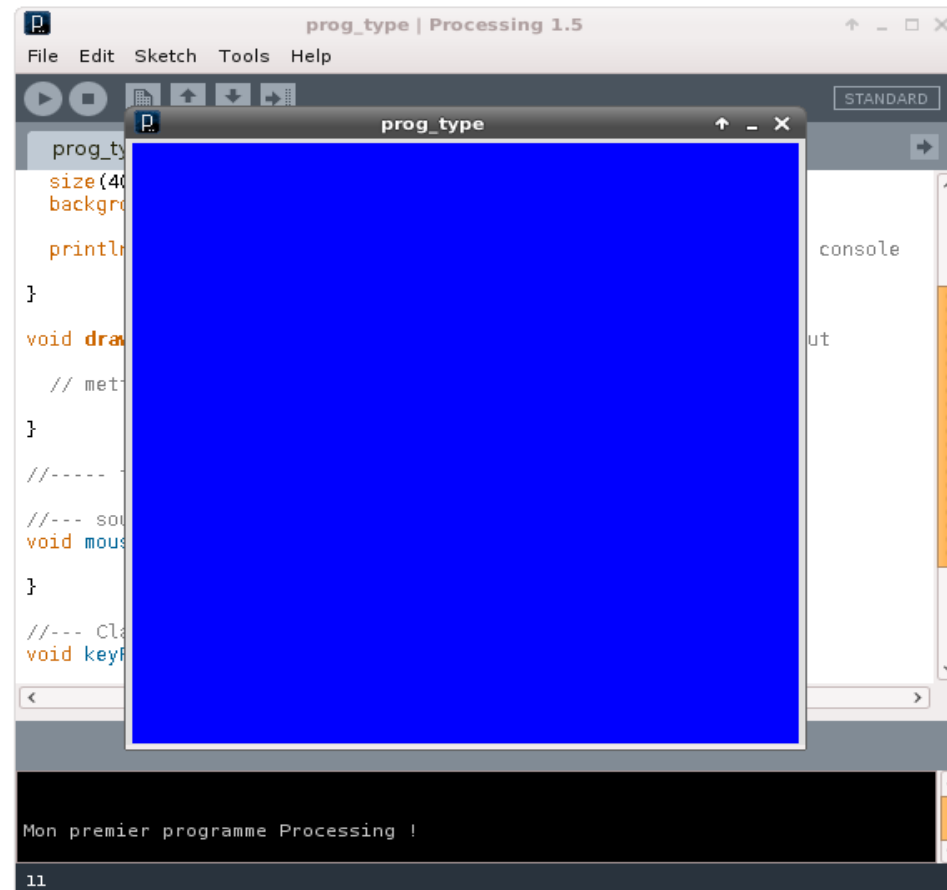
- Votre ordinateur dispose d'un clavier, d'une souris, d'un port série, etc... Le langage Processing pour vous permettre d'utiliser ces dispositifs utilise des fonctions « événements » qui sont appelées que lorsque cela est nécessaire.
- Par exemple, lorsque la souris est utilisée, la fonction **mousePressed()** est appelée lors d'un clic souris et exécute le code qu'on y a mis !

Utiliser la console

- Processing vous permet simplement d'afficher des messages dans la console de l'interface avec les fonctions... **print()** et **println()** !! Quand je vous avais dit que c'était simple...

// Programme Processing type

```
//-- entête déclarative --  
// déclarer ici les variables, objets, bibliothèques  
  
void setup() { // setup() est exécutée une fois  
  
    // mettre ici le code à exécuter au début  
    size(400,400); // fixe la taille de la fenêtre à 600x400 pixels  
    background(0,0,255); // fixe la couleur du fond à bleu (RGB)  
  
    println("Mon premier programme Processing !"); // affiche message  
    console  
  
}  
  
void draw(){ // draw est appelée à intervalle régulier - 60fps/défaut  
  
    // mettre ici le code à exécuter à chaque passage  
  
}  
  
//----- fonctions événements -----  
  
//--- souris ---  
void mousePressed() { // fonction appelée si bouton souris appuyé  
  
}  
  
//--- Clavier ---  
void keyPressed() { // si une touche est appuyée  
  
}  
  
} //--- fin si touche enfoncée
```

On obtient une fenêtre bleue de 400 x 400 pixels et le message s'est affiché dans la console.

Bravo !

Pour utiliser Processing avec Arduino, on peut se contenter de connaître uniquement quelques instructions Processing de base.

La documentation complète du langage Processing est ici : <http://processing.org/reference/>

17. Exemple avec Processing : allumer une LED à partir d'un clic à la souris

Voici donc à présent un premier exemple de programme Processing qui va interagir avec la carte Arduino par le port Série :

- cliquer sur le bouton droit de Souris dans la fenêtre Processing éteindra la LED
- cliquer sur le bouton gauche allumera la LED !

Voyons çà :

Entête déclarative

- on importe la librairie **Serial** du langage Processing qui va permettre de communiquer avec le port Série. C'est une différence avec le langage Arduino avec lequel la librairie Serial est incluse par défaut.

```
//-- entête déclarative --
```

```
// inclusion des librairies utilisées  
import processing.serial.*; // importe la librairie série processing
```

```
// déclaration objets  
Serial serialPort; // Création objet désignant le port série
```

Fonction **setup()**

Initialisation graphique initiale

- on initialise la fenêtre et la couleur du fond avec les instructions **size()** et **background()**

Initialisation du port Série utilisé

- on liste les ports série disponible la fonction **Serial.list()**
- on initialise la communication série avec le port [0] de la liste en initialisant l'objet serialPort à 115200.

L'initialisation de l'objet serialPort se fait selon une syntaxe d'initialisation d'objets propre au langage Java et se base sur un constructeur selon :
nomObjet = new Constructeur(param1, param2, param3, ...);
Constructeur a le même nom que la classe (ici Serial) et reçoit un ou plusieurs paramètres voire même aucun.

```
void setup() { // setup() est exécutée une fois
```

```
    // mettre ici le code à exécuter au début  
    size(200,200); // fixe la taille de la fenêtre  
    background(0,0,0); // fixe la couleur du fond à noir (RGB)
```

```
    //----- initialisation port série ----  
    println(Serial.list()); // affiche dans la console la liste des ports  
    séries  
    // Vérifier le bon port Serial.list()[index] est utilisé  
    serialPort = new Serial(this, Serial.list()[0], 115200); // Initialise  
    une nouvelle instance du port Série  
    // attention : le programme Arduino doit utiliser le même débit !  
}
```

Fonction **draw()**

- laissée vide : tout se passe dans la fonction de gestion de la souris.

```
void draw(){ // draw est appelée à intervalle régulier - 60fps/défaut  
    //rien ici - tout se passe dans mousePressed()  
}
```

Fonctions de gestion des évènements ()

Fonction de gestion de la souris

- Dans cette fonction on teste l'état du bouton de souris appuyé (droit ou gauche) :
 - si c'est le bouton gauche, on envoie la chaîne LED=ON sur le port série ce qui aura pour effet d'allumer la LED
 - si c'est le bouton droit, on envoie la chaîne LED=OFF sur le port série, ce qui aura pour effet d'éteindre la LED
- la fonction pour envoyer une chaîne sur le port série depuis Processing est de la forme `serialPort.write(« chaîne »)`;

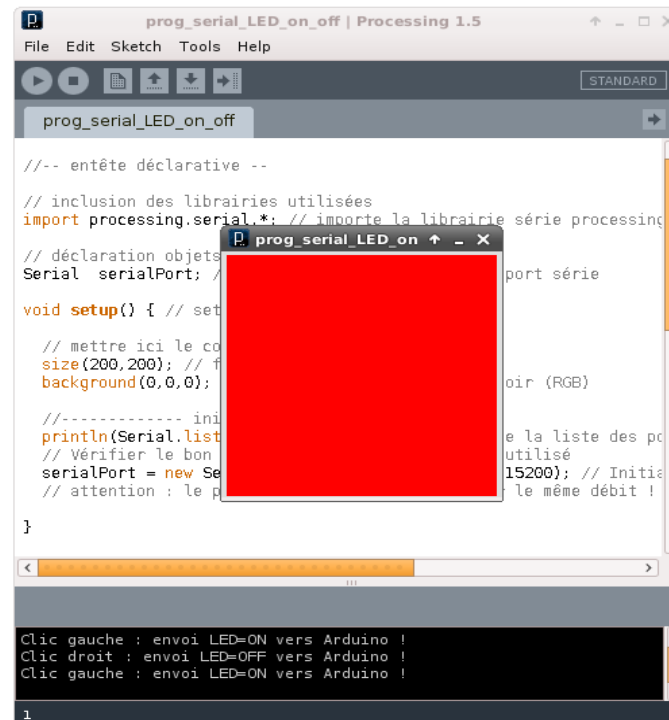
Noter le caractère spécial « \n » en fin de chaîne qui correspond au « saut de ligne » (carriage return / code ascii = 10)

La variable `mouseButton` et les identifiants `LEFT` et `RIGHT` utilisés ici sont internes au langage Processing et peuvent donc être utilisés directement.

```
//----- fonctions évènements -----  
  
//--- souris ---  
void mousePressed() { // fonction appelée si bouton souris appuyé  
  
    if (mouseButton == LEFT) { // test si appui bouton gauche  
  
        println("Clic gauche : envoi LED=ON vers Arduino !");  
        serialPort.write("LED=ON\n"); // envoie la chaîne suivi saut  
        // ligne sur le port Série  
        background(255,0,0); // fond en rouge  
  
    } // fin if LEFT  
  
    if (mouseButton == RIGHT) { // test si appui bouton droit  
  
        println("Clic droit : envoi LED=OFF vers Arduino !");  
        serialPort.write("LED=OFF\n"); // envoie la chaîne suivi  
        // saut ligne sur le port Série  
        background(0,0,0); // fond en noir  
  
    } // fin if RIGHT  
  
} // fin mousePressed
```

Fonctionnement du programme

- Vérifier que la carte Arduino est connectée et que le Terminal Série Arduino est fermé (sinon le port Série est déjà occupé !)
- Cliquer sur le bouton d'exécution : une fenêtre 200x200 apparaît et la liste des ports série s'affiche
- Cliquer sur la fenêtre avec le bouton gauche de la souris : elle change de couleur, un message s'affiche et la LED doit s'allumer
- Cliquer sur la fenêtre avec le bouton droit de la souris : elle change de couleur, un message s'affiche et la LED doit s'éteindre



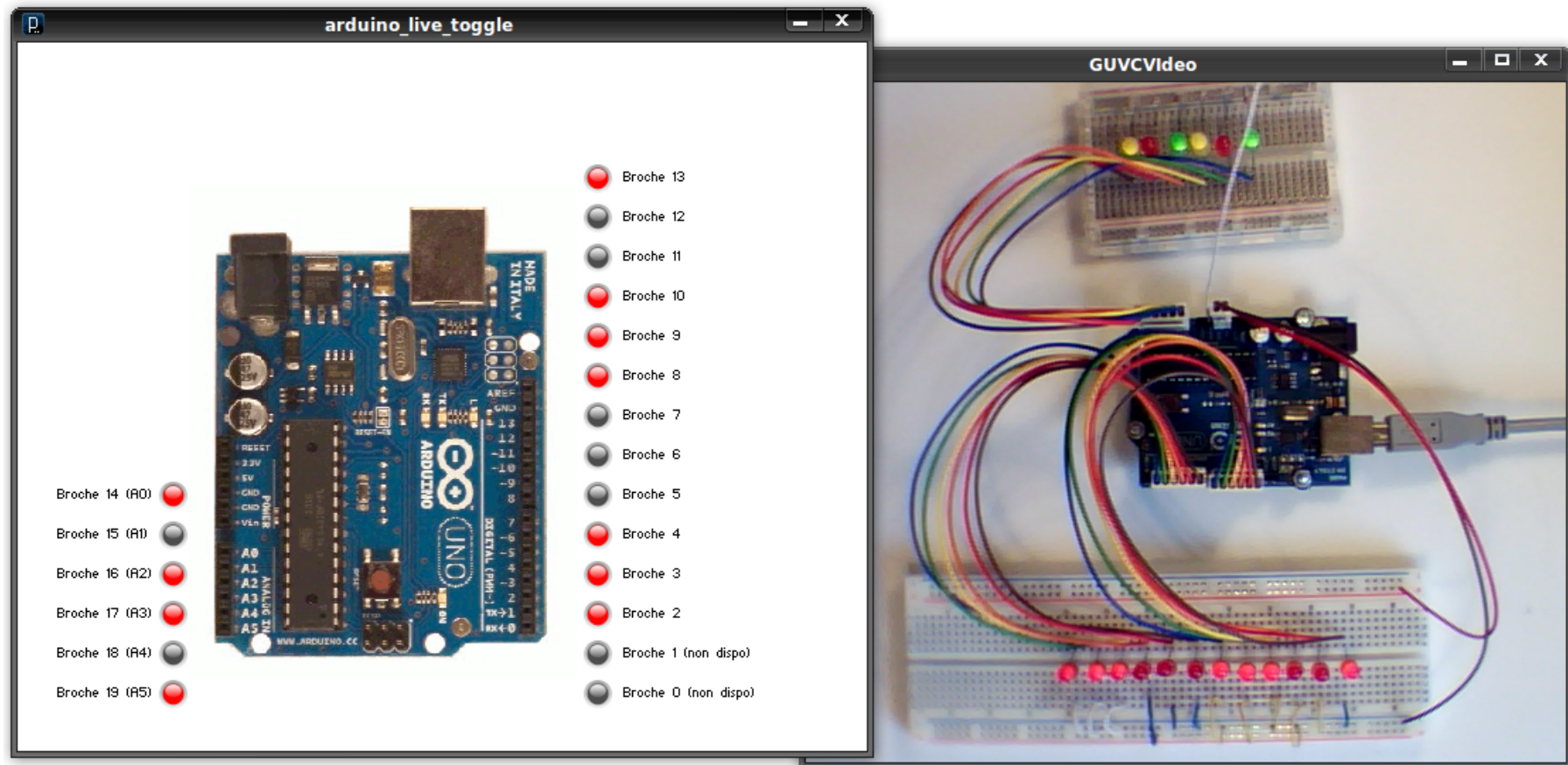
Vous avez ici les bases pour envisager le contrôle à partir du PC de toutes sortes de dispositifs à partir de la souris ou du clavier !

Remarquer également toute la souplesse de la mise au point : le programme peut tout d'abord être testé manuellement depuis le Terminal Série Arduino puis ensuite être utilisé avec une interface Processing côté PC.

On pourra de la même façon contrôler un dispositif sur un réseau local ethernet voire même internet.

18. Exemple avancé : Arduino Live ou comment contrôler toutes les broches Arduino depuis le PC !!

Je vous propose un programme Processing que j'ai écrit, très visuel, qui permet de contrôler toutes les broches de l'Arduino en mode ON/OFF à partir de boutons graphiques. Vous le trouverez ici (code Arduino et Processing + vidéo d'explication) : http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertSerieDepuisPCProcessingArduinoLive « Amusez-vous bien ! »



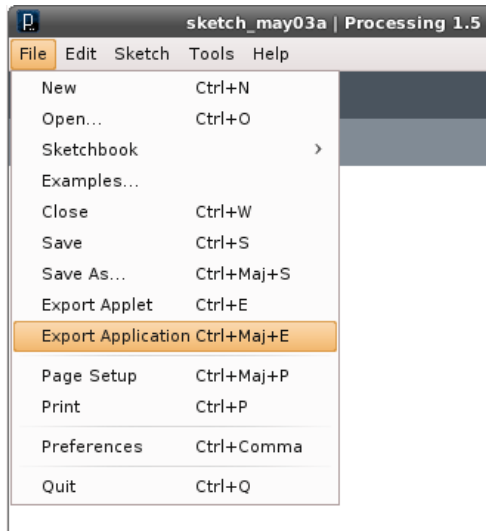
Purchased by Franck Ourion, franck.ourion@univ-lorraine.fr #6280170

Atelier Arduino : [Recevoir des nombres](#) en provenance du Terminal Série, contrôler la carte Arduino par le Terminal Serie et initiation à Processing.

19. Bon à savoir : On peut créer un exécutable à partir d'un programme Processing !

Il est facile de créer un exécutable à partir d'un programme Processing :

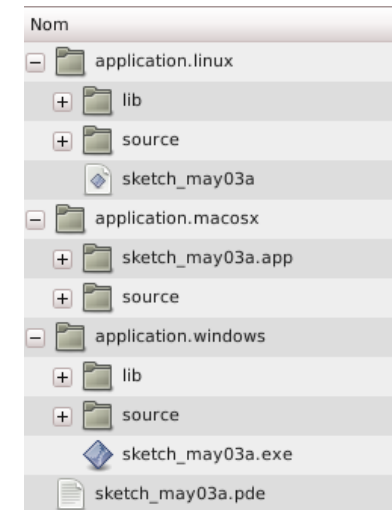
- aller dans fichier > export application



- dans la fenêtre qui s'ouvre, sélectionner les systèmes pour lesquels vous voulez un exécutable et choisissez vos options :



- Cliquer sur Export : le répertoire contenant les exécutables par système s'ouvre :



- A présent, il ne vous reste plus qu'à lancer votre application sur le système de votre choix. 3 clics... et c'est tout !

De façon semblable, on pourra créer un applet Java et même embarquer un programme Processing dans un navigateur Web (plus limité...)



Tout ceux qui se sont déjà frottés à la compilation d'une application utilisant le port série et à destination multi-OS comprendront immédiatement tout l'intérêt de Processing. Vous avez dit facile ?

20. *Réflexion autour du contrôle de l'Arduino par le PC*

- Permet de créer son propre « protocole » de communication très simplement
- Un même programme pourra d'abord être testé via le Terminal Série puis ensuite être utilisé avec une interface Arduino : très très pratique pour les développements et les mises au point !
- Remarquer également la simplicité avec laquelle il est possible de créer un véritable « périphérique USB » personnalisé, compatible multi-OS (Linux, Windows, Mac Os X) et son interface graphique côté PC. Ceux qui ont déjà essayé d'écrire un programme Java par exemple utilisant le port série comprendront de quoi je parle...

21. Les éléments du langage Arduino étudiés dans cet atelier

Structure

Variables et constantes

Fonctions

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

22. Les éléments du langage Processing abordés dans cet atelier

setup() | size() | background() | import | println | Serial | Serial.write() | mousePressed | mouseButton

La documentation complète du langage Processing est disponible ici :
<http://processing.org/reference>

23. *A présent, vous devriez être capable :*

- d'écrire un programme simple permettant d'afficher des messages sur le PC
- d'ouvrir et paramétrer le Terminal Série pour afficher des messages dans le logiciel 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 |
La fonction Serial.available() |
La fonction Serial.read() |
Rappel : Pour info : Le code ASCII (American Standard Code for Information Interchange) |
Rappel : Réglage et Utilisation du Terminal Série en « émission » vers Arduino |
Programme : Recevoir une valeur numérique entière sur le port Série |
|
Utile : Une fonction « clé en main » pour lire une valeur numérique + signe sur le port Série |
Contrôler une LED à partir du Terminal Série : le montage |
Contrôler une LED à partir du Terminal Série : le programme |
Fixer la fréquence de clignotement de la LED à partir du Terminal Série : le programme. |
Introduction à Processing |
Processing : le programme type |
Exemple avec Processing : allumer une LED à partir d'un clic à la souris |
Exemple avancé : Arduino Live ou comment contrôler toutes broches Arduino depuis le PC !! |
Bon à savoir : On peut créer un exécutable à partir d'un programme Processing ! |
Réflexion autour du contrôle de l'Arduino par le PC |
Les éléments du langage Arduino étudiés dans cet atelier |
Les éléments du langage Processing abordés 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