

Apprendre à **recevoir des chaînes de caractères** en provenance du Terminal Série avec la carte Arduino, à utiliser les **chaînes de caractères** avec l'objet String, à utiliser la boucle conditionnelle **while**.



## 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'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](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 comprendre le principe de communication du PC **vers** la carte Arduino
- découvrir les variables de type **char** (caractère)
- d'apprendre à utiliser les chaînes de caractères avec l'objet **String**
- découvrir l'utilisation de la boucle conditionnelle **while** et de l'instruction **break**

... afin d'être en mesure d'interagir avec la carte Arduino à partir du PC.

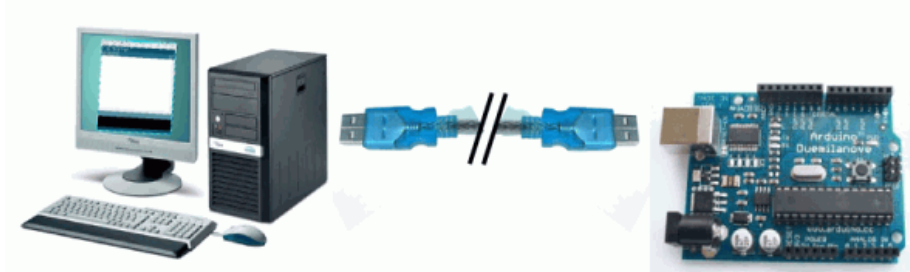


**Prêt ? C'est parti !**

## 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

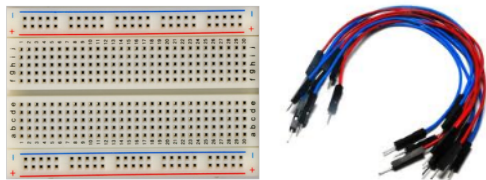


L'espace de développement Arduino associe :

- 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

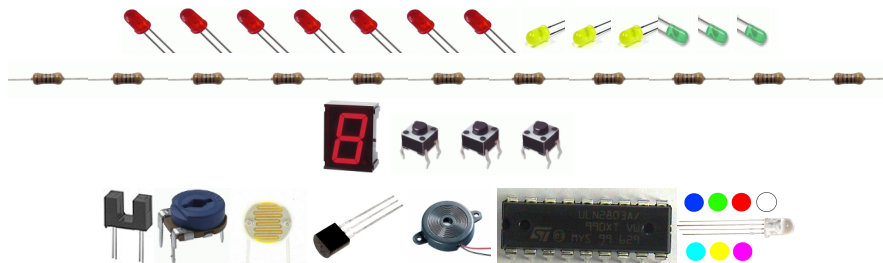


Pour réaliser des montages sans soudure, vous aurez besoin :

- 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) !

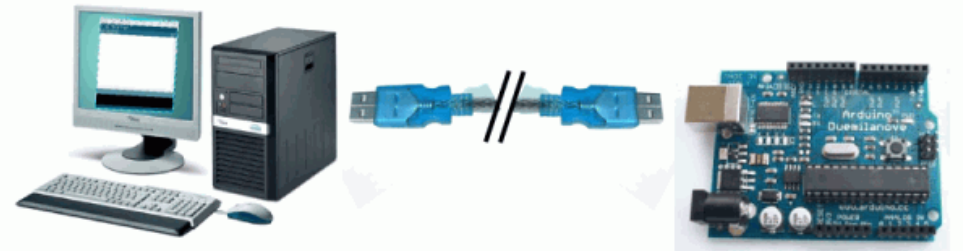
Le langage Arduino dispose de toutes les instructions nécessaires pour réaliser et programmer cette communication au sein d'un programme comme nous allons le voir ici.

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.

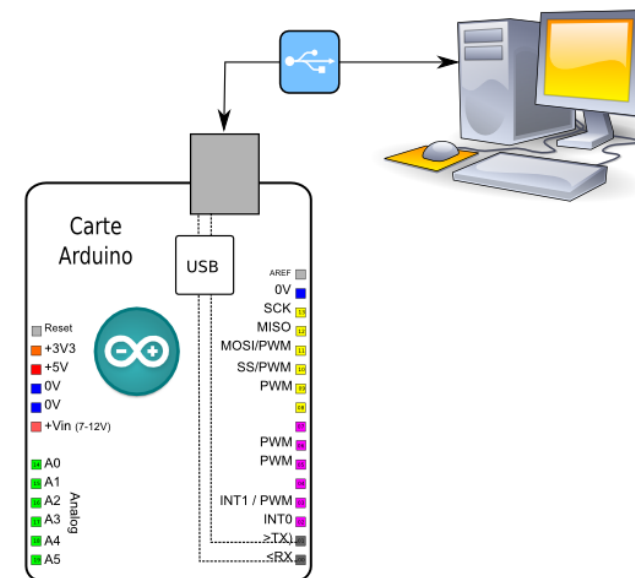
Au final, la carte Arduino programmée pourra fonctionner de 2 façons :

- 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.**

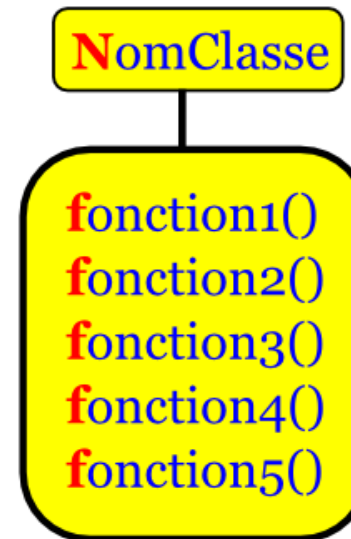
Tout comme une fonction, une classe aura un nom : pour distinguer une classe d'une fonction, **le nom d'une classe commencera par une MAJUSCULE.**

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

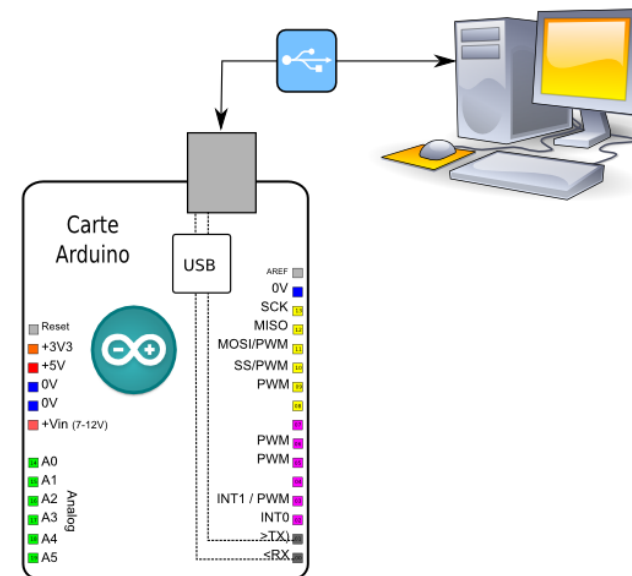
Ici, nous allons découvrir les fonctions de la classe Serial utiles en réception :

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





## 8. 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  
}
```

## 9. 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 :

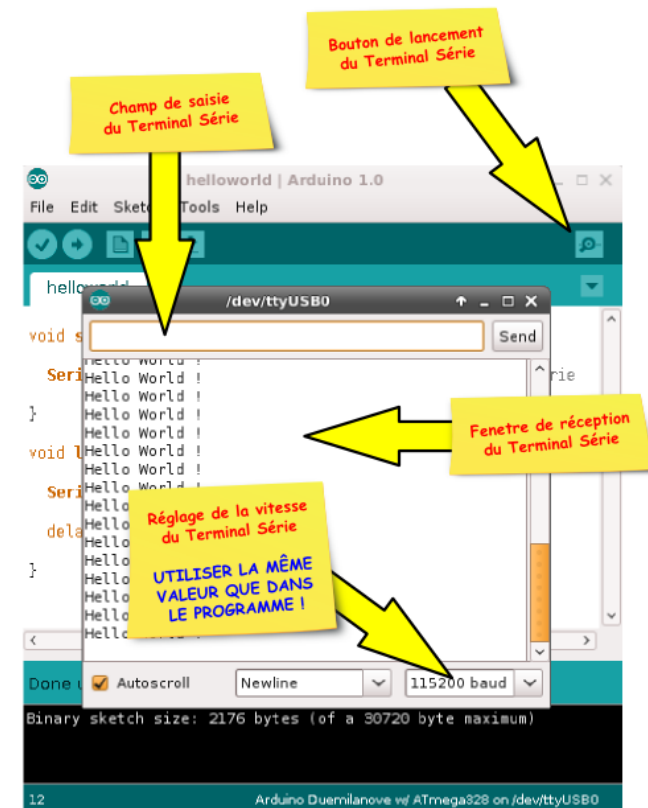
- soit Menu Tool > Serial Monitor
- soit clic sur le bouton Terminal Serie

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 ».



## 10. Langage : Type **int**, type **char** et code ASCII

Avant d'aborder la réception de données sur le port Série en provenance du PC, il est nécessaire de parler d'un sujet qui prête parfois à confusion...

### Quelques rappels au sujet des variables

En programmation, une « boîte mémoire » sur laquelle on colle une « étiquette » pour y mettre quelque chose : ça s'appelle une **variable** !

En programmation, la « taille » d'une « boîte mémoire » (ou variable) s'appelle le « **type** ». Pour dire les choses autrement, le type d'une variable, c'est sa catégorie, son genre .

### Le type **int**

Vous connaissez déjà le type **int** (pour integer = entier) qui peut contenir une **valeur entière** comprise entre -32536 et + 32535 : le int est un double octet (16 cases unitaires).

Le type int



### Le type **char**

Le langage Arduino dispose d'un type particulier qui va nous servir par la suite : le type **char**

Techniquement, une variable de type **char** va contenir également une valeur entière mais comprise entre -128 et + 127 : un char est donc un octet (8 cases unitaires).

Le type char



### Le type **char** représente un caractère !

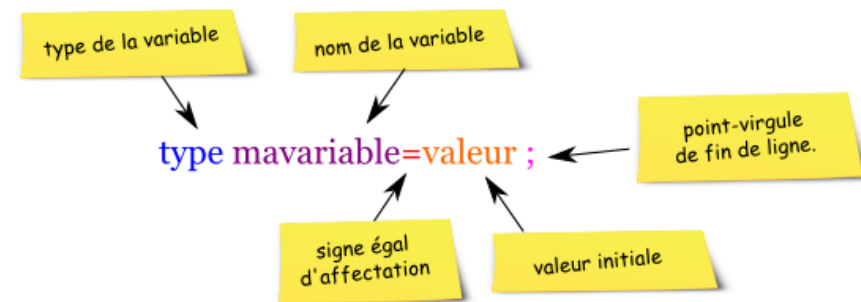
**Mais ATTENTION : une variable de type char sera interprétée en tant que code ASCII d'un caractère, pas en tant que valeur numérique !**

Par exemple, si un char contient la valeur 65, il sera considéré par le langage Arduino comme correspondant au caractère ayant le code ASCII = 65 (c'est à dire le A) et non pas comme la valeur 65...

### Déclaration d'une variable de type char

La déclaration d'une variable de type **char** se fait de la même façon que pour n'importe quel autre type de variable :

#### Déclaration d'une variable avec initialisation.



Deux syntaxes sont possibles

**char** maVariable=65; // à partir du caractère ASCII – ici lettre A

**char** maVariable='A'; // à partir du caractère lui-même entre ''

### Les types **int**, **char** et les fonctions **print()** ou **println()**

Soit le char myChar=65 et le int myInt=65 :

**Serial.println**(myChar); // affiche la lettre A (code ASCII = 65)

**Serial.println**(myInt); // affiche le nombre 65 (valeur numérique)

### Convertir un **int** en **char** : la fonction **char()**

**Serial.println**(char(myInt)); // affiche la lettre A (code ASCII = 65)

## 11. Pour info : Le code ASCII (American Standard Code for Information Interchange)

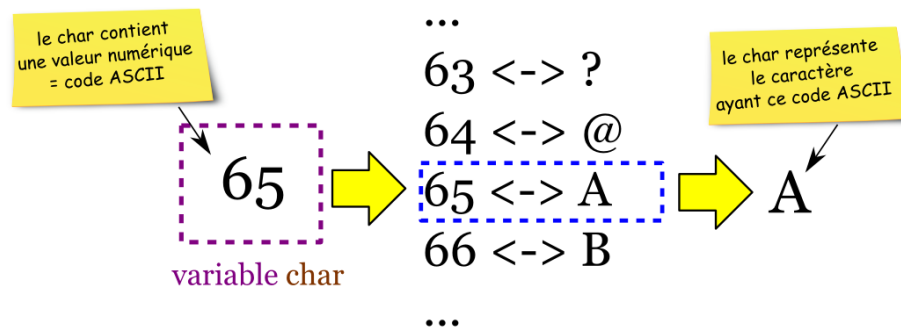
### Le code ASCII

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

## 12. Recevoir un caractère sur le port Série et l'afficher dans le Terminal : le programme

A présent, on va donc écrire notre premier programme qui va recevoir un caractère sur le port Série. Prêt ? C'est parti !

### Entete déclarative

On va déclarer à ce niveau :

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

### 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 simple condition **if** pour tester la présence d'un octet dans la file d'attente du port série avec la fonction **Serial.available()**
- Si un octet est reçu, on affiche successivement :
  - sa valeur numérique (c'est à dire le code ASCII du caractère reçu)
  - puis le caractère correspondant

### 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 « No Line Ending » pour aucun ajout après la chaîne saisie.

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

int octetReception=0; // variable de réception octet
char caractereReception=0; // variable de réception caractère

//--- 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

} // fin de la fonction setup()

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

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

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

        caractereReception=char(octetReception); // récupere le caractere à
partir du code Ascii

        Serial.print("Arduino a reçu valeur :");
        Serial.print(octetReception); // affiche la valeur de l'octet
        Serial.print(" (code ASCII) correspondant au caractere ");
        Serial.println(caractereReception); // affiche le caractere

    } // fin if

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

## 13. Réglage et Utilisation du Terminal Série 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 défilante :
  - 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

Amusez-vous à modifier le mode d'émission pour voir les différences...





## 14. Langage : La classe String : une classe pour gérer facilement les chaînes de caractères !

Une fois que l'on sait utiliser une variable contenant un caractère, on a logiquement envie de pouvoir manipuler des chaînes de caractères.

### Utiliser un tableau de variables **char** ?

La solution qui vient à l'esprit est de vouloir créer un tableau de variables de type char, ce qui est effectivement une solution possible, mais qui présente un gros inconvénient : la taille du tableau ne pourra pas varier au cours du programme ! Dès lors impossible d'ajouter simplement des caractères à moins de créer d'emblée un grand tableau...

### Une bien meilleure solution : utiliser la classe String !

La lourdeur de la gestion des chaînes de caractères à l'aide d'un tableau a poussé l'équipe du langage Arduino à créer une classe spéciale dédiée à la gestion des chaînes de caractères : c'est un des domaines où toute la puissance du langage Arduino est évidente !

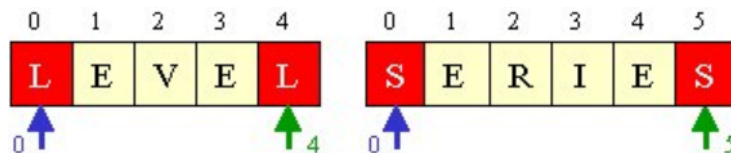
### La classe String : un conteneur de chaîne de caractères !

La classe String c'est tout d'abord un « **conteneur** » de **chaîne de caractères de taille variable** : pas besoin d'en préciser la taille initiale et on pourra à tout moment lui ajouter un caractère supplémentaire.

### La classe String : de nombreuses fonction très pratiques !

Mais ce n'est pas tout : la classe String va mettre à notre disposition toute une **série de fonctions très très pratiques** pour utiliser les chaînes de caractères : ajouter une chaîne, connaître la taille de la chaîne, se positionner à un endroit précis de la chaîne, éliminer les espaces, extraire une sous-chaîne, etc...

En un mot, **souplesse maximale** !



### Déclaration d'un objet de la classe String

Un objet de la classe String se déclare tout simplement de la même façon qu'une variable sous la forme :

```
String chaineReception=""; // déclare un objet String vide
```

Usage avancé : noter que l'on ne parle plus ici de variable, mais d'objet car on crée une instance (l'objet créé) de la classe String (le moule). Le concept d'objet est plus large que celui de variable et associe un espace mémoire ainsi que des fonctions spécifiques. En pratique, on peut considérer qu'un objet String est une sorte de « super-variable ».

### Opérations de base sur d'un objet String

On peut ajouter très simplement une chaîne de caractère ou des Strings entre eux :

```
maChaine=maChaine+ «test »; // ajoute une chaîne au String
```

```
maChaine=maChaine+ maChaine2; // ajoute une chaîne au String
```

### Quelques fonctions de la classe String

- **charAt()** : renvoie le caractère à la position voulue
- **compareTo()** : compare le String à une chaîne
- **endsWith()** : teste si le String se termine par une sous-chaîne
- **indexOf()** : recherche la position d'une chaîne
- **length()** : renvoie la longueur de la chaîne
- **setCharAt()** : modifie le caractère à la position donnée
- **startsWith()** : teste si le String commence par une sous-chaîne
- **substring()** : extrait une sous-chaîne à la position donnée
- **trim()** : élimine les espaces

### Objet String et les fonctions **print()** ou **println()**

Un objet String s'utilise très simplement avec les fonctions **print()** et **println()** :

```
Serial.print(maChaine); // affiche le String
```



## 15. Pour info : toutes les façons valides d'initialiser un String

```
String stringOne = "Hello String";           // en utilisant une chaîne de caractères
String stringOne = String('a');               // conversion d'un caractère simple en objet String appelé stringOne
String stringTwo = String("This is a string"); // conversion d'une chaîne de caractère en objet String appelé
stringTwo
String stringOne = String(stringTwo + " with more"); // concaténation d'un objet String et d'une chaîne
String stringOne = String(13);                 // conversion d'un nombre en base 10 par défaut
String stringOne = String(analogRead(0), DEC); // conversion d'une valeur int en base 10
String stringOne = String(45, HEX);            // conversion de la valeur 45 en base hexadécimale
String stringOne = String(255, BIN);           // conversion de la valeur 255 en base binaire
String stringOne = String(millis(), DEC);      // conversion d'une valeur long en base 10
```

**Pas de panique !**

**Vous n'avez pas besoin de retenir toutes les fonctions de la classe String par coeur : nous les utiliserons au fur et à mesure des besoins !**  
**Retenez simplement que c'est un outil très puissant pour manipuler des chaînes de caractères !**

## 16. Programme : Stocker dans une chaîne les caractères reçus sur le port Série et l'afficher dans le Terminal

A ce stade, on va pouvoir stocker tous les caractères reçus sur le port série dans une chaîne de caractères de taille variable : un objet String.

### Entete déclarative

On va déclarer à ce niveau :

- on déclare une variable **int** pour stocker l'octet en réception (code ASCII du caractère).
- on déclare une variable **char** pour stocker le caractère correspondant.
- on déclare un objet **String** vide pour stocker la chaîne de caractère

### 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 simple condition **if** pour tester la présence d'un octet dans la file d'attente du port série avec la fonction **Serial.available()**
- Si un octet est reçu :
  - on récupère le caractère correspondant à l'aide de la fonction de conversion **char()**
  - on ajoute la caractère à l'objet **String** puis 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 « No Line Ending »** pour aucun ajout après la chaîne saisie.

```
//--- 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

//--- 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

} // fin de la fonction setup()

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

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

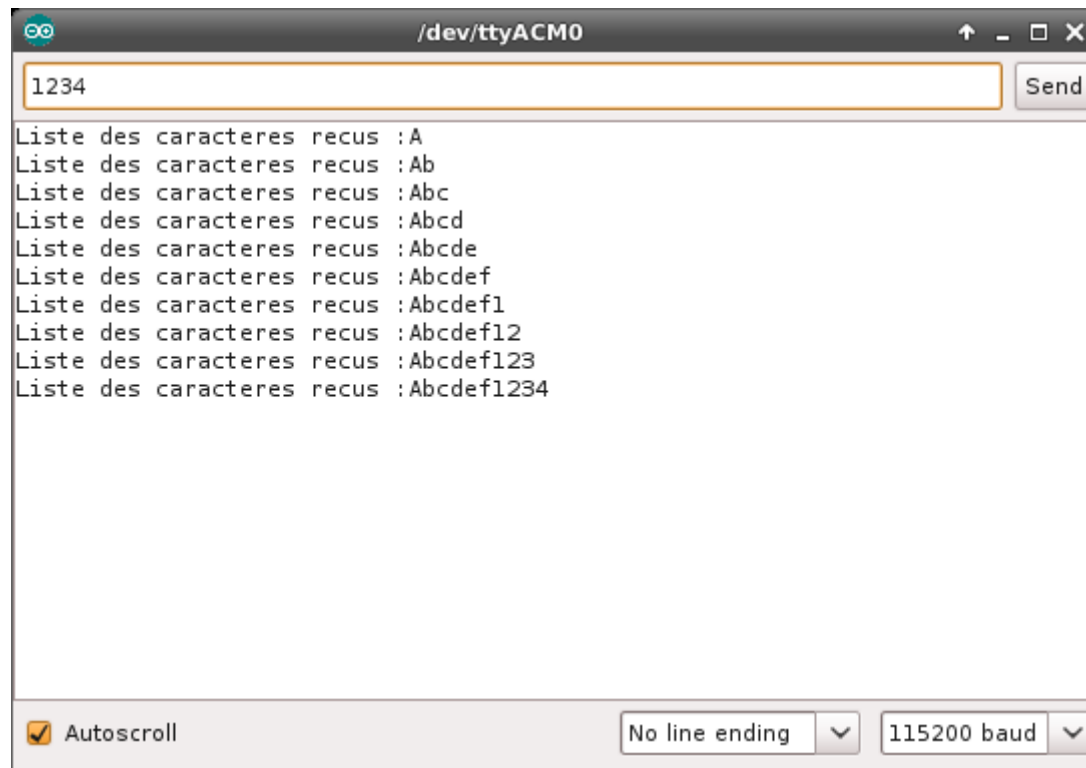
        octetReception=Serial.read(); // lit le 1er octet de la file
        d'attente

        caractereReception=char(octetReception); // récupère le caractère à
        partir du code Ascii

        chaineReception=chaineReception+caractereReception; // ajoute la
        caractère au String
        Serial.print("Liste des caracteres recus :");
        Serial.println(chaineReception);

    } // fin if

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



## 17. Langage : La boucle conditionnelle **while( )**

Dans certaines situations, il peut être utile d'effectuer une opération **tant qu'une certaine condition est vraie**. Par exemple lors de la réception de caractères sur le port série (tu me vois venir ?... non, sans blague ?!) ou d'un appui sur un bouton poussoir. La solution passe par ce que l'on pourrait appeler une boucle conditionnelle qui dit au microprocesseur :

- **tant que** (=while en anglais) telle condition est vraie, alors faire ceci
- sinon faire cela.

Cette boucle conditionnelle :

- est intéressante car elle permet, à la différence d'un boucle **for**, de **répéter une tâche un nombre fois non défini à l'avance**, répétition qui dépend d'une condition,
- est **à utiliser avec parcimonie, car elle peut bloquer un programme** (si la condition est toujours vraie) mais cet effet est parfois intéressant pour une exécution pas à pas ou un arrêt à un endroit précis du code.

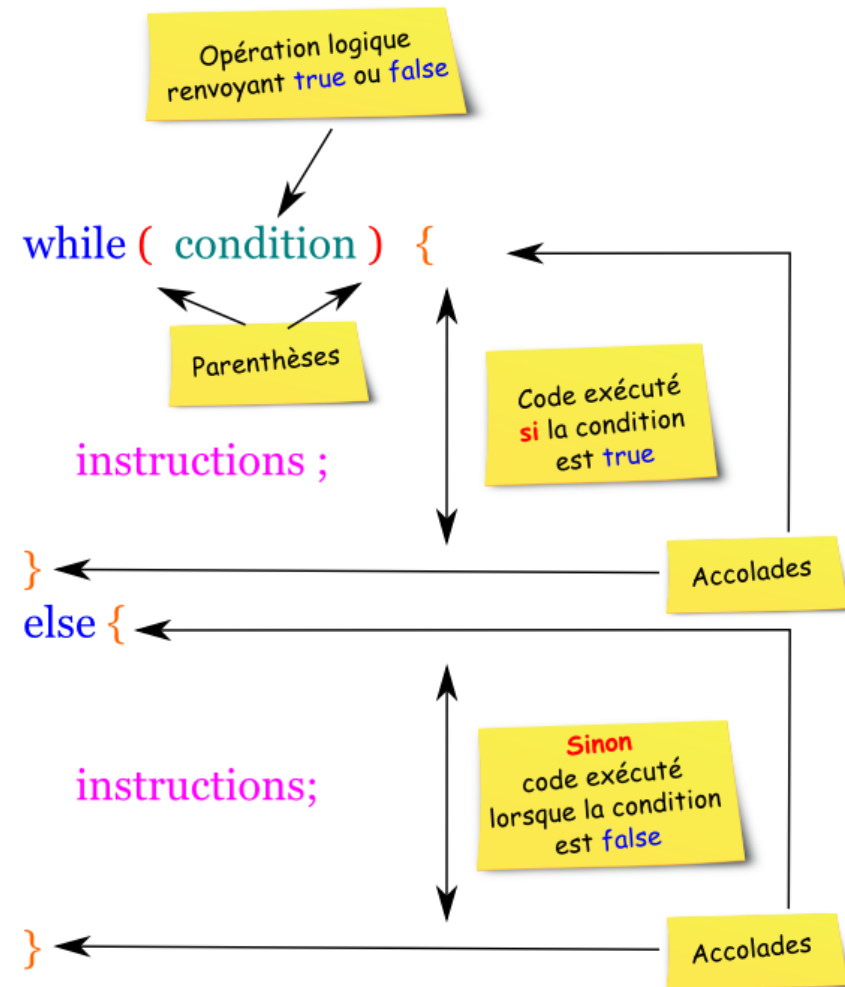
L'instruction utilisée pour une boucle conditionnelle « **tant que** » est l'instruction **while ... else...** (tant que... sinon...) que l'on écrit de la façon suivante (presque comme une condition **if...else...**):

- on commence par le mot clé **while** suivi de **la condition entre ( )**
- suivi des **{ } qui contiennent les instructions** à exécuter si la condition est vraie, chaque instruction devant être suivie d'un ;
- en option, on peut compléter du mot clé **else** suivi des **{ }** qui contiennent les instructions à exécuter si la condition est fausse.

**La condition devra être une opération logique :**

- qui renverra une valeur de type boolean soit **true** (vrai) soit **false** (faux)
- on utilisera pour cela les **opérateurs logiques de comparaison**
- pour info, **0** est considéré comme **false** et toute valeur différente de **0** est considérée comme **true** (ainsi, **if(1)** est toujours vrai !)

**Truc :** **while(1);** permet de réaliser un point d'arrêt à n'importe quel endroit d'un programme (la condition est toujours vraie). Parfois utile !



## 18. Rappel : les opérateurs logiques et leur utilisation (utilisables avec *if*, *while*, *else*, etc..)

La condition de base utilisable est de tester si une variable vaut une certaine valeur :

- on écrira la **condition sous la forme `variable==valeur` (2 signes == )**
- Par exemple on écrira `if ( variable==2) { instructions ;}` ce qui veut dire « si la condition « variable vaut 2 » est vraie alors exécuter les instructions »

**Ne pas confondre l'opération logique de test d'égalité == avec le signe = d'affectation :**  
c'est une erreur fréquente de débutant et même de programmeur expérimenté... !!

### Les opérateurs logiques de comparaison

- `x == y` : VRAI si x est **égal à** y
- `x != y` : VRAI si x est **différent de** y
- `x < y` : VRAI si x est **inférieur à** y
- `x > y` : VRAI si x est **supérieur à** y
- `x <= y` : VRAI si x est **inférieur ou égal à** y
- `x >= y` : VRAI si x est **supérieur ou égal à** y

### Les opérateurs booléens (permettent d'enchaîner des conditions entre-elles)

- **&&** (ET logique) : VRAI seulement si les deux conditions sont VRAI

```
if (digitalRead(2) == HIGH && digitalRead(3) == HIGH) { // lit l'état de 2 boutons poussoirs
// ...
}
```

- **||** (OU logique) : VRAI si l'une des deux conditions est VRAI

```
if (x > 0 || y > 0) { // si x supérieur à 0 ou si y supérieur à 0
// ...
}
```

- **!** (NON logique) : VRAI si l'opérande est FAUX – cas particulier : `!x` est VRAI chaque fois que `x=0` (« tordu » mais pratique !)

```
if (!x) {
// ...
}
```

## 19. Pour info : une variante : la boucle conditionnelle **do... while**

Juste pour votre information, car en pratique ça ne sert pas souvent, il existe une variante de la boucle `while()` : la boucle **do... while()** (« faire... tant que.. »).

### Description

La boucle **do** / **while** ("faire tant que" en anglais) fonctionne de la même façon que la boucle **while**, à la différence près que la condition est testée à la fin de la boucle, et par conséquent la boucle **do** sera toujours exécutée au moins une fois.

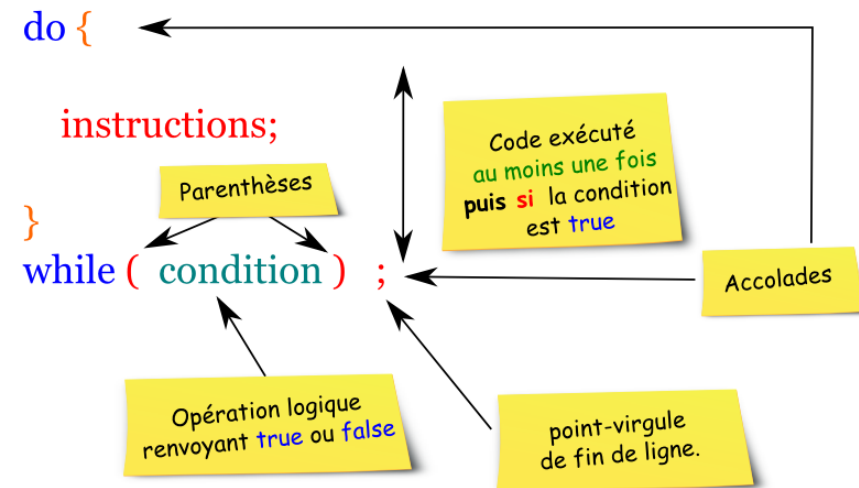
C'est subtil, mais ça peut parfois servir...

### Syntaxe

```
do // faire...
{
    // bloc d'instruction
} while (condition); // tant que la condition est vraie
```

### Exemple

```
do // faire...
{
    delay(50);          // attendre la stabilisation du capteur
    x = readSensors(); // lit la valeur de la tension du capteur
} while (x < 100); // ...tant que x est inférieur à 100
```



## 20. Programme : Recevoir une chaîne de caractères sur le port Série et l'afficher dans le Terminal

Si vous êtes attentif, vous avez du remarquer que le programme précédent recevait les caractères 1 à la fois, à chaque passage dans `loop()`. Il serait plus pratique de pouvoir recevoir une chaîne de caractère d'un seul coup, « tant que des caractères sont disponibles »... et on va donc utiliser pour cela une boucle **while** ! (ben oui, je vous apprends des trucs qui vont servir !)

### Entete déclarative

On va déclarer à ce niveau :

- on déclare une variable **int** pour stocker l'octet en réception (code ASCII du caractère).
- on déclare une variable **char** pour stocker le caractère correspondant.
- on déclare un objet **String** vide pour stocker la chaîne de caractère

### 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 cette fois 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 est présent : on récupère le caractère correspondant à l'aide de la fonction de conversion **char()**, puis on ajoute la caractère à l'objet **String**. On réalise une petite pause entre 2 réceptions.
- 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 « No Line Ending »** pour aucun ajout après la chaîne saisie.

```
//--- 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

//--- 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

} // fin de la fonction setup()

//--- 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

        caractereReception=char(octetReception); // récupère le caractère à
        partir du code Ascii

        chaineReception=chaineReception+caractereReception; // ajoute la
        caractère au String

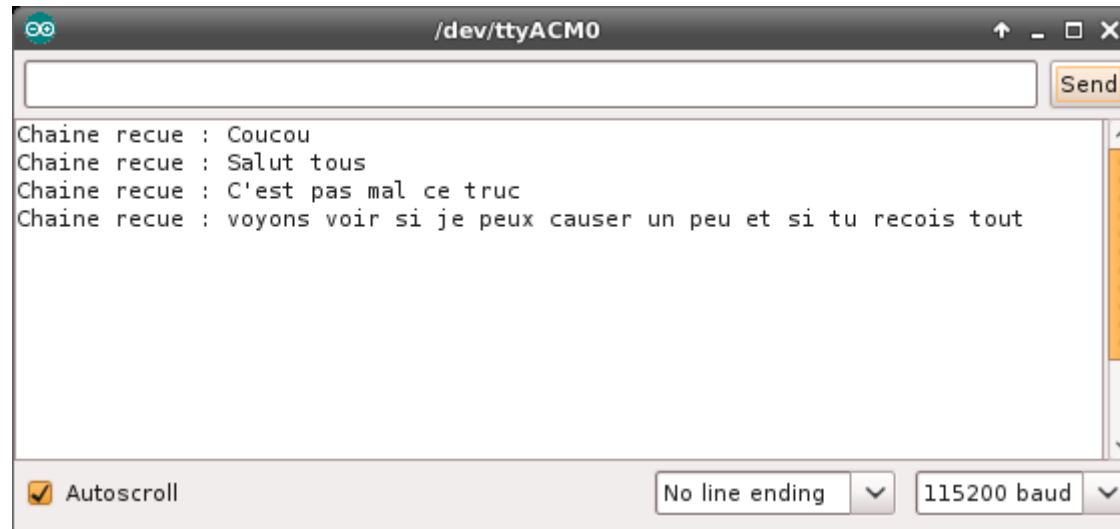
        delay(1); // laisse le temps au caractères d'arriver

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

    //----- une fois la chaîne reçue
    if (chaineReception!="") { // la chaîne n'est pas vide
        Serial.print("Chaîne recue : ");
        Serial.println(chaineReception);
        chaineReception=""; // Vide la chaîne
    } // fin if

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





## 21. Langage : l'instruction **break**

L'instruction **break** permet de sortir d'une boucle même si son déroulement n'est pas terminé !

### Description

- L'instruction **break** est utilisée pour sortir d'une boucle **do**, **for** ou **while**, en passant outre le déroulement normal de la boucle.
- Pour info, cette instruction est également utilisée pour sortir d'une instruction **switch** (mais on n'a pas encore vu ça) .

### Syntaxe

- **Attention : l'instruction break ne nécessite pas de parenthèses vides** (c'est une exception... ah là là... « pourquoi y z'aurait pas pu faire simple.... ! »...
- Mettre par contre, le point virgule comme d'hab' !

**break;**

### Exemple

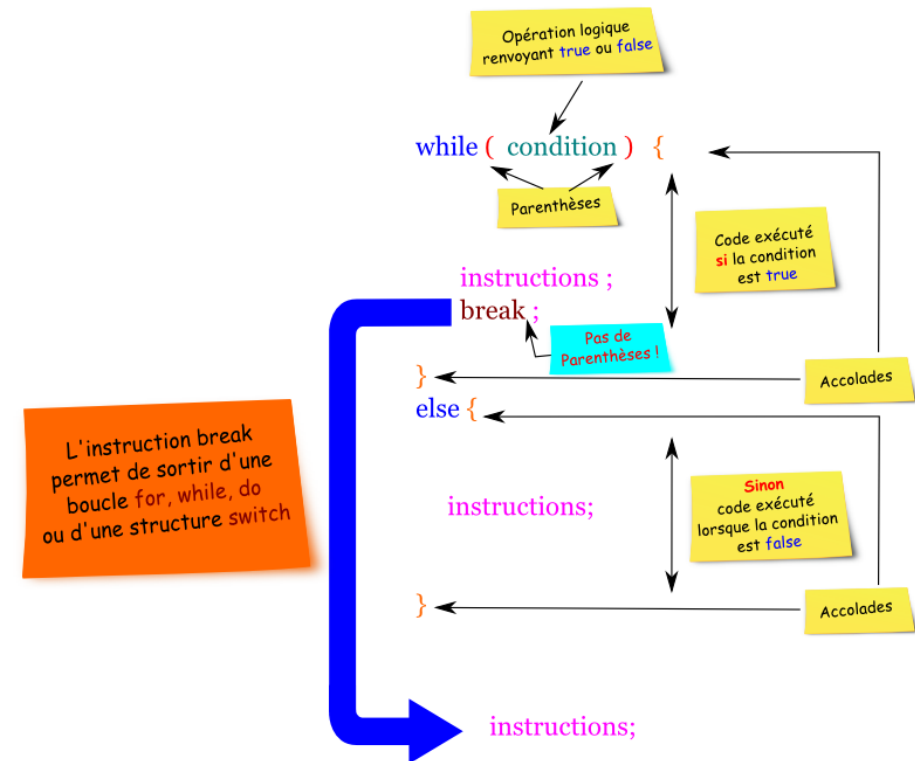
```
for (x = 0; x < 255; x++) // boucle for comptant x de 0 à 255
{
    // mettre une impulsion de largeur x sur la broche
    digitalWrite(PWMPin, x);

    // lire la valeur de la tension d'un capteur sur la broche
    sens = analogRead(sensorPin);

    // si la mesure est supérieure à un seuil, on sort de la boucle
    if (sens > threshold){
        x = 0;
        break; // sortie de la boucle
    }

    delay(50); // pause de 50ms
}
```

**L'instruction break n'est pas utilisable avec une condition if else !**



## 22. Programme : Recevoir une chaîne de caractères suivie d'un saut de ligne sur le port Série et l'afficher

Améliorons encore un peu les choses... Lorsque l'on va envoyer des chaînes de caractères vers Arduino, c'est pour lui donner des instructions... Imaginons que l'on en envoie plusieurs à la suite : le port série contiendra « fairececi fairecela puis ceci puis cela »... Il faudrait que chaque instruction soit bien séparée des autres : le truc va consister à intercaler un « saut de ligne » (caractère ascii =10) entre chaque chaîne donnant : « fairececi » puis « fairecela » puis « puisceci » puis « puiscela »... allez action !

### 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, un objet **String** vide pour stocker la chaîne de caractère

### 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
String chaineReception=""; // déclare un objet String vide

//--- 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

} // fin de la fonction setup()

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

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

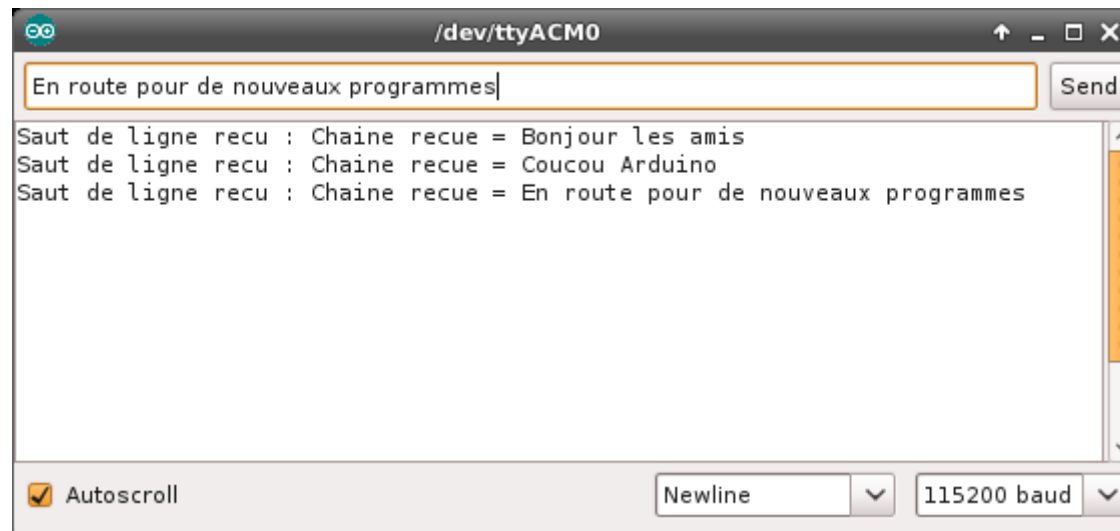
        octetReception=Serial.read(); // lit le 1er octet de la file
        d'attente

        if (octetReception==10) { // si Octet reçu est le saut de ligne
            Serial.print ("Saut de ligne reçu : ");
            Serial.println ("Chaîne recue = "+chaineReception); // affiche
la chaîne recue
            chaineReception=""; //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); // récupère le caractère
à partir du code Ascii
            chaineReception=chaineReception+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()
```



Faites la même chose en réutilisant l'option No Line Ending et voyez le résultat :  
aucune chaîne ne sera prise en compte tant que le saut de ligne n'est pas arrivé !  
Rebasculer sur l'option « NewLine » et cliquer sur « send » : la chaîne est prise en compte !

De cette façon, les chaînes peuvent arriver rapidement ou lentement sur le port Série,  
tant que le saut de ligne n'est pas reçu, elle n'est pas prise en compte.  
Ceci donne de la « robustesse » à la communication série du PC vers Arduino.

## 23. 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)

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

- d'écrire un programme capable de recevoir des chaînes de caractères 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 |  
La fonction Serial.available() |  
La fonction Serial.read() |  
Rappel : « Hello world ! » : programme Arduino envoyant un message vers le PC via le port USB |  
Lancer et paramétrer le Terminal Série pour afficher/recevoir des messages entre le PC et Arduino |  
Langage : Type int, type char et code ASCII |  
Pour info : Le code ASCII (American Standard Code for Information Interchange) |  
Recevoir un caractère sur le port Série et l'afficher dans le Terminal : le programme |  
Réglage et Utilisation du Terminal Série en réception |  
Langage : La classe String : une classe pour gérer facilement les chaînes de caractères ! |  
Pour info : toutes les façons valides d'initialiser un String |  
Programme : Stocker dans une chaîne les caractères reçus sur le port Série et l'afficher dans le Terminal |  
Langage : La boucle conditionnelle while( ) |  
Rappel : les opérateurs logiques et leur utilisation (utilisables avec if, while, else, etc..) |  
Pour info : une variante : la boucle conditionnelle do... while |  
Programme : Recevoir une chaîne de caractères sur le port Série et l'afficher dans le Terminal |  
Langage : l'instruction break |  
Programme : Recevoir une chaîne de caractères suivie d'un saut de ligne sur le port Série et l'afficher |  
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)