5. Programmation des Systèmes Intelligents et Connectés avec Arduino

2016-2017
Ecole Supérieure Polytechnique / Département Génie Informatique

Les premiers outils de prototypage des systèmes électroniques sont certes puissants, mais nécessitent un processus de développement long et difficile à apprendre, cela requiert de solides connaissances en programmation informatique et en électronique.

Salué pour avoir établi le mouvement des “makers” et générer un regain d’intérêt pour l’électronique, Arduino a considérablement évolué depuis que la première carte a été mise en vente il y a un peu plus de dix ans. Arduino a permis à plus de gens d’apprendre, d’expérimenter et de réaliser des projets complets qui auraient auparavant exigés du matériel dédié et coûteux. Arduino a suscitée rapidement l’intérêt des ingénieurs professionnels. En effet, ces cartes bon marché leur permettent de valider une conception de projet et fabriquer rapidement un prototype.

❑ Arduino : présentation

Arduino est une carte basée sur un microcontrôleur abordable et simple à mettre en œuvre pour développer des montages électroniques numériques programmables à base de microprocesseur. Le système Arduino comprend à la fois :

- → **Le développement matériel de sa carte,**
- → **Mais aussi le développement de son environnement de programmation.**

Alors que la première génération de cartes incorporait seulement un microcontrôleur AVR d'Atmel, on note aujourd'hui l'intégration à Arduino de plusieurs autres types de cartes issues de autres projets telles que les célèbres cartes ESP8266.

❑ Arduino : une plate-forme matérielle

L'élément de base de la carte Arduino est le microcontrôleur, ce qui explique que chaque version d'Arduino a un type de microcontrôleur bien distinct. La version la plus populaire étant sans nulle doute la UNO. Dans la suite, nous travaillerons avec :

→ Arduino Uno R3

→ Arduino Mega 2560

❑ Arduino UNO

La carte Arduino Uno est une carte à microcontrôleur basée sur l'ATmega328. Elle contient tout ce qui est nécessaire pour le fonctionnement du microcontrôleur. Son interfaçage avec l'ordinateur se fait via un port USB, qui sert également de source d'alimentation.

❑ Arduino UNO : caractéristiques

Le tableau suivant présente les principales caractéristiques de l'Arduino Uno.

ATmega328	Processeur AVR 8 bits à architecture RISC
	16 MHz de vitesse d'horloge
	32 Ko de mémoire FLASH
	2 Ko de mémoire SRAM
	1 Ko d'EEPROM
	Timers/Counters : Timer0 et Timer2 (8 bits), Timer1 (16bits)
Alimentation	5V (port USB) ou 7–12V (alimentation externe)
	AREF : tension de référence pour les entrées analogiques
	GND : c'est la masse
	Sources de tension de +5V
	Sources de tension de +3.3V
	Vin : alimentation non stabilisée
Connectique	14 Entrées/Sorties numériques (5V, 40mA, 50MΩ) numéroté de 0 à 13
	6 Entrées analogiques (10bits) numérotées de A0 à A5

❑ Arduino MEGA

La carte Arduino Mega 2560 est une carte à microcontrôleur basée sur l'ATmega2560. C'est la carte qu'il faut si on manque de broches E/S et de mémoire avec la carte Uno.

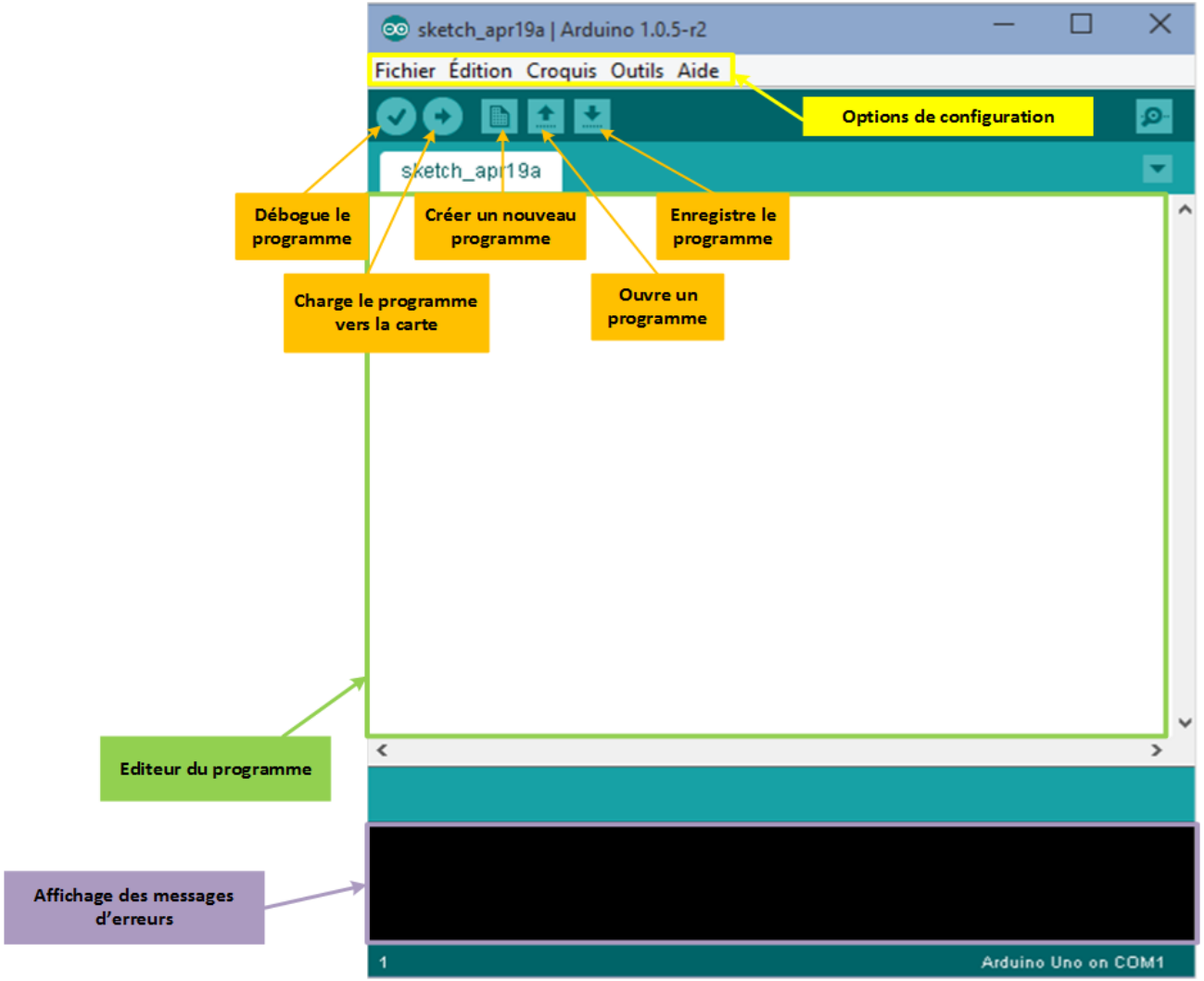
❑ Arduino MEGA : caractéristiques

Le tableau suivant présente les principales caractéristiques de Arduino Mega

ATmega2560	Processeur AVR 8 bits à architecture RISC
	16 MHz de vitesse d'horloge
	256 Ko de mémoire FLASH
	8 Ko de mémoire SRAM
	4 Ko d'EEPROM
	Timers/Counters : Timer0 et Timer2 (8 bits), Timer1 (16bits)
Alimentation	5V (port USB) ou 7−12V (alimentation externe)
	AREF : tension de référence pour les entrées analogiques
	GND : c'est la masse
	Sources de tension de +5V
	Sources de tension de +3.3V
	Vin : alimentation non stabilisée
Connectique	54 Entrées/Sorties numériques (5V, 40mA, 50MΩ) numéroté de 0 à 53
	16 Entrées analogiques (10bits) numérotées de A0 à A15

❑ Arduino : un environnement logiciel

Le logiciel Arduino est gratuit, open source et multiplateformes: Windows, Linux, Mac OS

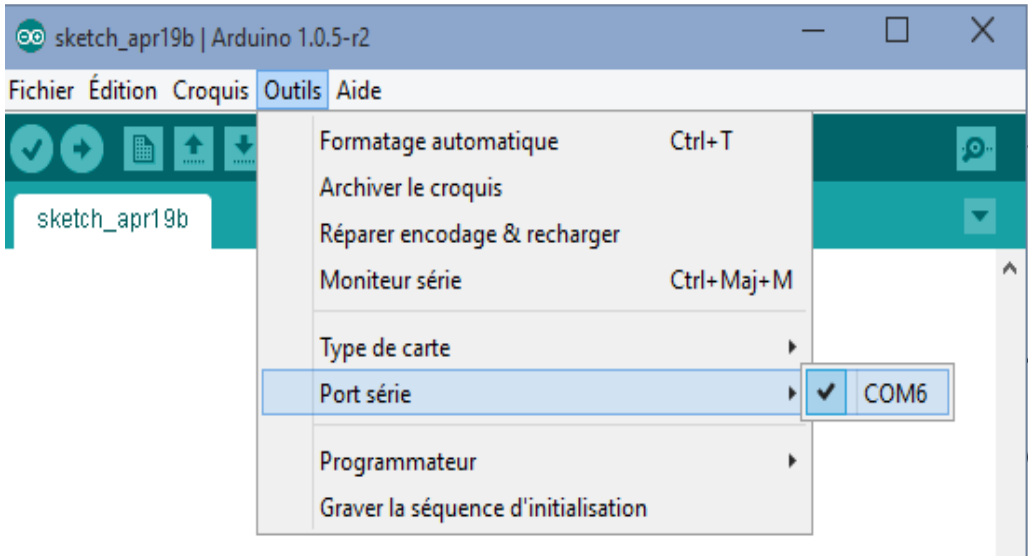


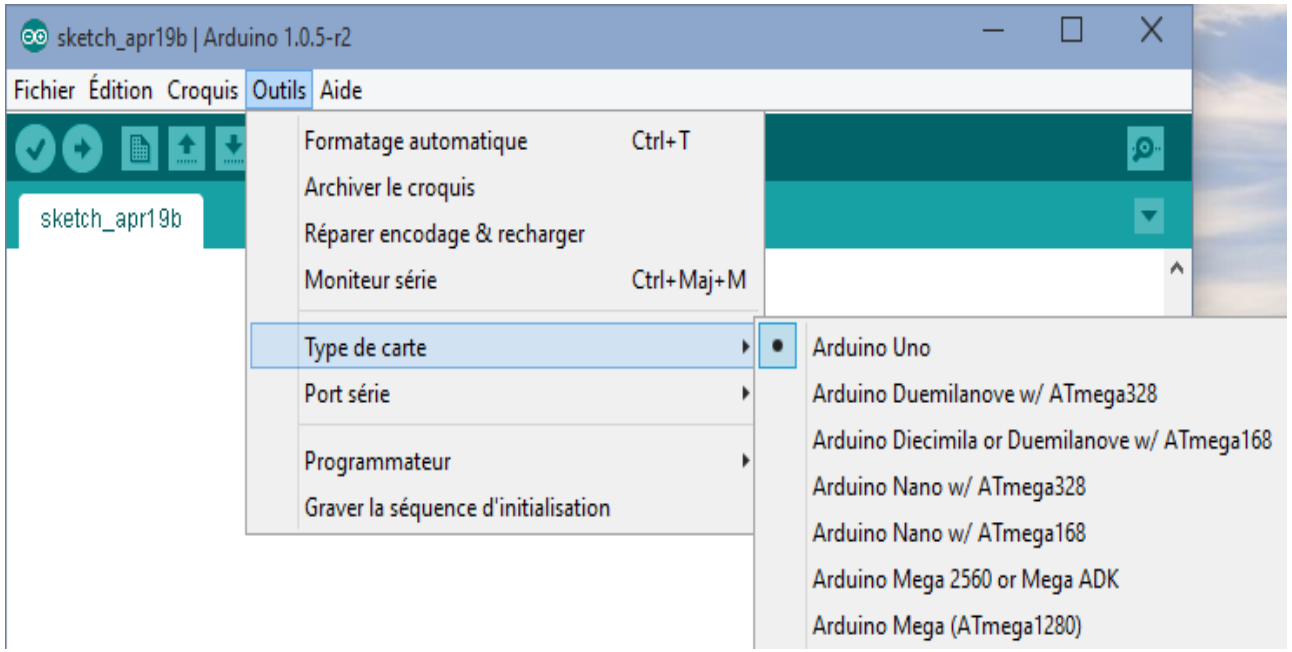
❑ Arduino : configuration

Avant de pouvoir utiliser Arduino, une configuration préalable du logiciel s'impose.

→ **Sélection du port série :** indiquer le numéro de port sur lequel Arduino est connecté à l'ordinateur.

→ **Sélection du port série :** préciser au logiciel avec quel type de carte on va travailler





❑ Introduction au langage Arduino

Le langage Arduino est basé sur les langages C/C++ et supporte toutes les constructions standards du langage C et quelques-uns des outils du C++. Le langage Arduino repose sur l'utilisation du compilateur C pour les microcontrôleurs AVR, AVR Libc, et vous permet d'utiliser la plupart de ses fonctions.

❑ Structure du programme Arduino

La structure d'un programme doit toujours comporter les fonctions `setup()` et `loop()`

sketch_apr20a §

```
// Définition des constantes, variables globales
// directives de compilation define, include, etc.

void setup() {
  // initialisation des ressources de la carte,
  // configuration des entrées/sorties,
  // définition de la vitesse de fonctionnement du port série, etc.
  // setup() n'est exécuté qu'une seule fois.
}

void loop() {
  // les instructions contenues ici sont exécutées indéfiniment en boucle
  // Seule une coupure de l'alimentation de la carte ou un appui sur le bouton Reset
  // permet de quitter le programme.
}
```

❑ Syntaxe de base

Comme tout langage de programmation, il existe des règles de syntaxes à respecter.

→ **Point-virgule :** marque la fin d'une instruction, on peut séparer plusieurs instructions

```
int a = 13;
digitalWrite(pin1,HIGH), digitalWrite(pin2,LOW), digitalWrite(pin3,HIGH);
```

→ **#include :** utilisée pour insérer des librairies externes dans le programme

```
#include <dht11.h>
```

→ **Commentaires :** sont ignorées par le compilateur

```
// Commentaire sur une seule ligne
/* Commentaire multi-lignes
   Suite commentaire
*/
```

❑ Opérateurs arithmétiques

Arduino possède tous les opérateurs de bases pour réaliser des opérations arithmétiques.

→ Opérateur d'assignement =

```
int sensVal;     // declare une variable entière de 16 bits nommée senVal
senVal = analogRead(0);     // mémorise la valeur passée au CAN à la variable
```

→ Addition + , Soustraction - , Multiplication * , Division / et Modulo %

```
result = value1 + value2;
result = value1 - value2;
result = value1 * value2;
result = value1 / value2;
result = value1 % value2;
```

La variable à gauche de l'opérateur d'assignement (le signe =) nécessite d'être capable de stocker la valeur qu'on lui attribue.

❑ Les autres opérateurs

En plus des opérateurs arithmétiques, le langage Arduino présente d'autres opérateurs pour plus de facilité de programmation.

→ Opérateurs de comparaison

```
x == y      // x est égal à y
x != y      // x est différent de y
x < y       // x est inférieur à y
x > y       // x est supérieur à y
x <= y      // x est inférieur ou égal à y
x >= y      // x est supérieur ou égal à y
```

→ Opérateurs logiques booléens

```
&&      // ET logique
||      // OU logique
!       // NON logique
```

❑ Structures de contrôle

Les structures de contrôle sont basées sur des conditions et des boucles.

→ Condition if

```
if (uneVariable > 50)
{
   // faire quelque chose
}
```

→ Conditions if/else

```
if (brocheCinqEntree < 500) {
   // faire l'action A
}
else if (brocheCinqEntree < 1000) {
   // faire l'action B
}
else {
   // faire l'action C
}
```

❑ Structures de contrôle

→ Boucle while

```
while(expression) {  // tant que l'expression est vraie
   // instructions à effectuer
}
```

→ Boucle for

```
for (initialisation; condition; incrementation) {
   //instruction à exécuter
}
```

→ L'instruction switch / case

```
switch (var) { // début de la structure
    case 1:
      //faire quelque chose quand la variable est égale à 1
    break;
    case 2:
      //faire quelque chose quand la variable est égale à 2
    break;
    default:
      // si aucune condition n'est vraie, le code par défaut sera exécuté
}
```

❑ Les variables

Les variables sont utilisées pour stocker des valeurs. Avant d'être utilisée, toutes les variables doivent être déclarées.

```
int inputVariable1;
long inputVariable2 = 0;      // les 2 façons sont correctes
```

→ Types de variables

Type de variable	Type de nombre stocké	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
int	entier	-32'768 à +32'767	16 bits	2 octets
long	entier	-2'147'483'648 à +2'147'483'647	32 bits	4 octets
char	entier	-128 à +127	8 bits	1 octet
float	décimale	-3.4×10^{38} à $+3.4 \times 10^{38}$	32 bits	4 octets
double	décimale	-3.4×10^{38} à $+3.4 \times 10^{38}$	32 bits	4 octets

Pour les types non signés, on repère ces types par le mot unsigned qui les précède.

❑ Conversion des types de données

Arduino permet de faire la conversion d'un type de donnée vers un autre.

```
byte(x)   // Convertit une valeur en donnée de type byte
int(x)    // Convertit une valeur en donnée de type int
long(x)   // Convertit une valeur en donnée de type long
float(x)  // Convertit une valeur en donnée de type float
String(x) // Convertit une valeur en en donnée de type string
```

❑ Les constantes Arduino prédéfinies

Les constantes prédéfinies du langage Arduino sont des valeurs particulières ayant une signification spécifique.

→ Définition de niveaux logiques

Les états VRAI et FAUX dans Arduino sont représentées par : `true` et `false`.

→ Définition des niveaux de broche

Une broche numérique ne peut prendre/être qu'à deux valeurs : `HIGH` ou `LOW`.

- `HIGH`

En mode *INPUT*, Arduino renverra HIGH si une tension de plus 3V est présente sur la broche.
En mode *OUTPUT*, Arduino met la broche sur 5V avec une intensité maximale de 40mA.

- `LOW`

En mode *INPUT*, Arduino renverra LOW si une tension de moins de 2V est présente sur la broche.
En mode *OUTPUT*, Arduino met la broche sur 0 volts.

❑ Les chaînes de caractères

Les chaînes de caractères sont représentées sous forme de tableau de variables de type char et se termine par un zéro. Toutes les expressions suivantes sont des déclarations valides pour des chaînes de caractère.

```
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
char Str4[ ] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino";
```

→ **Concaténation de chaine :** la concaténation de chaine se fait grâce à l'opérateur **+**

```
int temp = 25;
char message[ ] = "Alerte temperature : "+String(temp)+"C";
```

❑ Les fonctions

Sous Arduino, il existe trois catégories de fonctions :

→ **Les fonctions crées :** sont définies dans le code source du programme

```
void nomFunction(datatype argument){ // ouverture de la fonction
   // vos instructions ici
} // fermeture de la fonction
```

→ **Les fonctions prédéfinies :** Arduino regorge de plusieurs fonctions permettant de traiter les entrées/sorties numériques et analogiques.

→ **Les fonctions issues de bibliothèques :** cette catégorie de fonctions nécessite que la bibliothèque contenant la fonction soit importée.

```
#include <dht11.h>
dht11 DHT;        // création d'un objet de type dht11
DHT.read(2);      // lecture des données situées au pin 2: température et humidité
```

❑ Configuration des Entrées/Sorties numériques

Configure la broche spécifiée pour qu'elle se comporte soit en entrée, soit en sortie. La fonction prend deux paramètres :

→ **`pinMode(pin, mode)`**

- pin : le numéro de la broche de la carte Arduino que l'on veut configurer
- mode : soit *INPUT* ou *OUTPUT*

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

void setup()
{
   pinMode(ledPin, OUTPUT);  // met la broche numérique en sortie
}
```

❑ Ecrire sur des Entrées/Sorties numériques

Met un niveau logique HIGH ou LOW sur une broche numérique, sa tension est mise à 5V pour le niveau HIGH, 0V pour le niveau LOW.

→ **digitalWrite(pin, value)**

```
void loop()
{
   digitalWrite(ledPin, HIGH);    // allume ou éteint la LED selon le montag
}
```

❑ Lire sur des Entrées/Sorties numériques

Lit l'état d'une broche précise en entrée numérique, et renvoie la valeur HIGH ou LOW.

→ **digitalRead(pin)**

```
void loop()
{
    val = digitalRead(boutonPin);    //  lit l'état de la broche en entrée
}
```

❑ Lire sur des entrées analogiques

Revoie toujours une **valeur numérique** comprise entre **0** et **1023**.. Ce dernier est connecté à un convertisseur analogique-numérique 10 bits de tension de référence 5V.

→ **`analogRead(pin)`**

```
void loop()
{
    val = analogRead(thermoPin); // lit la valeur numérique codée en décimale
}
```

Il existe une autre fonction capable de trouver la valeur analogique correspondante.

→ **`map(val, fromLow, fromHigh, toLow, toHigh)`**

- val : valeur numérique, comprise entre fromLow et fromHigh
- fromLow = 0
- fromHigh = 1023
- toLow = 0
- toHigh = 5000 (mV)

❑ Générer un signal analogique sur une sortie numérique

Consiste à générer une impulsion de largeur et période voulue (PWM - Pulse Width Modulation). La broche générera une onde carrée stable avec un rapport cyclique de longueur spécifiée.

→ **`analogWrite(pin, valeur)`**

- pin : devra être une broche ayant la fonction PWM, sur la UNO ça pourra être une des broches 3, 5 ,6 ,9 ,10 ou 11.
- valeur : la largeur du rapport cyclique entre 0 et 255

```
void loop()
{
   analogWrite(pin, 128); // crée un signal de rapport cyclique 128 cad 50%
}
```

Il n'est pas nécessaire de faire appel à l'instruction pinMode() pour mettre la broche en mode OUTPUT.

❑ Gestion du temps

Consiste

❑ Arduino et la communication série UART

La communication série est indispensable pour dialoguer avec une carte Arduino puisque c'est le mode de transmission utilisé par défaut.

- → **Hardware :** toutes les cartes Arduino disposent d'au moins d'un port série. Pour l'Arduino UNO, ses ports UART se trouvent aux broches numériques pin0 (RX) et pin1 (TX). Si on utilise cette fonctionnalité, on ne peut pas utiliser les broches 0 et 1 en tant que broches E/S.
- → **Software :** Arduino dispose d'une bibliothèque *Serial* pour la communication série. Cette bibliothèque met à la disposition du programmeur de nombreuses fonctions permettant d'exploiter tous les avantages de la communication série.

❑ Initialiser une communication série

Pour initialiser une communication, on utilise la fonction begin qui prend deux paramètres dont l'un est optionnel. L'émetteur et le récepteur doivent utiliser la même configuration.

→ **`Serial.begin(speed)`**

- speed : fixe le débit de communication (en bauds) pour la communication série. Les débits les plus couramment utilisés sont : 9600 et 115200.
- config : paramètre optionnel, permet de spécifier la taille des données, le bit de parité et les bits stop. La valeur par défaut est SERIAL_8N1.

```
Serial.begin(9600);
Serial.begin(9600, SERIAL_8N1);
```

❑ Envoi de données texte

Permet d'envoyer du texte sous format ASCII. Cette fonction prend deux paramètres dont l'un est optionnel.

→ **`Serial.print(val)`**

- val : la valeur à afficher. N'importe quel type de données.
- format : optionnel, spécifie la base utilisée (pour les nombres entiers : BIN, DEC, HEX, OCT ou le nombre de décimales (pour les nombres de type float).

```
Serial.print("hello world!");
Serial.print(val, format);
```

La fonction **`println`** fonctionne de la même manière que print à la différence que la fonction println insère à la fin des données un caractère de "retour de chariot" (ASCII 13, ou '\r') et un caractère de "nouvelle ligne" (ASCII 10, ou '\n').

❑ Envoi de données binaires

Toutes les données ne peuvent pas être encodées sous format ASCII. Il est parfois plus aisé d'envoyer directement les données binaires sur le port série.

→ **`Serial.write(val)`**

- val : une valeur à envoyer sous forme d'octet simple
- buf : un tableau pour envoyer une série d'octets
- len : la taille du tableau

```
Serial.write(val);
Serial.write(buf, len);
```

❑ Réception de données

On peut vérifier le nombre d'octets présents dans le buffer du port série grâce à la fonction available.

→ **`Serial.available()`**

- cette fonction est surtout combinée avec la condition if pour tester s'il reste des données au niveau du buffer.

La fonction *read* renvoie le premier octet de donnée (0−255) conservé dans le buffer du port série, ou renvoie −1 si aucune donnée n'est disponible.

→ **`Serial.read()`**

- La valeur retournée est de type int

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

❑ Exercices d'applications

Proposer un programme les applications suivantes :

① Faire clignoter une LED.

② Une sonnerie à buzzer commandée par un bouton poussoir.

③ Un capteur de luminosité à base de photorésistance capable d'afficher les valeurs de mesures sur le moniteur série.

④ Un capteur de température et d'humidité à base du DHT11 capable d'afficher les valeurs de mesures sur le moniteur série.