Sommaire

1. Introduction

2. Outil de développement Arduino

3. Outil de développement IDE d'Arduino

4. Le langage Arduino

5. Quelques cas d'applications

Outil de Développement Arduino

I. Introduction

1. Qu'est-ce qu'Arduino?

Le système Arduino est un outil pour fabriquer de petits ordinateurs qui peuvent capter et contrôler davantage de choses du monde matériel que votre ordinateur de bureau. C'est une *plateforme open-source* d'électronique programmée qui est basée sur une simple carte à microcontrôleur (de la famille AVR), et un logiciel, véritable environnement de développement intégré, pour écrire, compiler et transférer le programme vers la carte à microcontrôleur.

Arduino peut être utilisé pour développer des objets interactifs, pouvant recevoir des entrées d'une grande variété d'interrupteurs ou de capteurs, et pouvant contrôler une grande variété de lumières, moteurs ou toutes autres sorties matérielles. Les projets Arduino peuvent être autonomes, ou bien ils peuvent communiquer avec des logiciels tournant sur votre ordinateur (tels que Flash, Processing ou MaxMSP). Les cartes électroniques peuvent être fabriquées manuellement ou bien être achetées pré-assemblées; le logiciel de développement open-source peut être téléchargé gratuitement.

Le langage de programmation Arduino est une implémentation de Wiring, une plateforme de développement similaire, qui est basée sur l'environnement multimédia de programmation Processing.

2. Pourquoi Arduino ?

Il y a de nombreux microcontrôleurs et de nombreuses plateformes basées sur des microcontrôleurs disponibles pour l'électronique programmée. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, et beaucoup d'autres qui offrent des fonctionnalités comparables. Tous ces outils prennent en charge les détails compliqués de la programmation des microcontôleurs et les intègrent dans une présentation facile à utiliser. De la même façon, le système Arduino simplifie la façon de travailler avec les microcontrôleurs, tout en offrant plusieurs avantages pour les enseignants, les étudiants et les amateurs intéressés par les autres systèmes :

- **Pas cher** : les cartes Arduino sont relativement peu coûteuses comparativement aux autres plateformes. La moins chère des versions du module Arduino peut être assemblée à la main, et même les cartes Arduino pré-assemblées coûtent moins de 25 Euros (microcontrôleur inclus...) !!!

- **Multi-plateforme** : Le logiciel Arduino, écrit en Java, tourne sous les systèmes d'exploitation Windows, Macintosh et Linux. La plupart des systèmes à microcontrôleurs sont limités à Windows.
- **Un environnement de programmation clair et simple**: L'environnement de programmation Arduino (= le logiciel Arduino) est facile à utiliser pour les débutants, tout en étant assez flexible pour que les utilisateurs avancés puisse en tirer profit également. Pour les enseignants, il est basé sur l'environnement de programmation Processing : les étudiants qui apprennent à programmer dans cet environnement seront déjà familiarisés avec l'aspect du logiciel Arduino.
- **Logiciel Open Source et extensible** : Le logiciel Arduino et le langage Arduino sont publiés sous licence open source, disponible pour être complété par des programmateurs expérimentés. Le langage peut être aussi étendu à l'aide de librairies C++, et les personnes qui veulent comprendre les détails techniques peuvent reconstruire le passage du langage Arduino au langage C pour microcontrôleur AVR sur lequel il est basé. De la même façon, vous pouvez ajouter du code du langage AVR-C directement dans vos programmes Arduino si vous voulez.

 Matériel Open source et extensible : Les cartes Arduino sont basé sur les microcontrôleurs Atmel ATMEGA8, ATMEGA168, ATMEGA 328, etc... Les schémas des modules sont publiés sous une licence Creative Commons, et les concepteurs de circuits expérimentés peuvent réaliser leur propre version des cartes Arduino, en les complétant et en les améliorant. Même les utilisateurs relativement inexpérimentés peuvent fabriquer la version sur plaque d'essai de la carte Arduino, dans le but de comprendre comment elle fonctionne et pour économiser de l'argent.

3. Domaine d'utilisation

- Physical computing : Au sens large, construire des systèmes physiques interactifs qui utilisent des logiciels et du matériel pouvant s'interfacer avec des capteurs et des actionneurs.
- Électronique industrielle et embarquée
- Art / Spectacle
- Domotique
- Robotique Modélisme DIY , Hacker, Prototypage,
- Education, Etc.

II. Matériel Arduino

Il existe plusieurs versions de cartes Arduino. La version courante de base, la Duemilanove (2009 en Italien - les créateurs de l'Arduino sont des Italiens...), utilise le microcontrôleur Atmel ATmega328. La version précédente Diecimila, et les premières versions de la Duemilanove utilisent le microcontrôleur ATmega168, tandis que les cartes plus anciennes utilisent le microcontrôleur ATmega8. La carte Arduino Mega est basée sur l'ATmega1280.

Il y a trois types de cartes :

- Les dites « officielles » qui sont fabriquées en Italie par le fabricant officiel : Smart Projects
- Les dits « compatibles » qui ne sont pas fabriqués par Smart Projects, mais qui sont totalement compatibles avec les Arduino officielles.
- Les « autres » fabriquées par diverse entreprise et commercialisées sous un nom différent (Freeduino, Seeduino,)

III. Les différentes cartes

Des cartes Arduino il en existe beaucoup ! Peut-être une centaine toutes différentes ! Je vais vous montrer lesquelles on peut utiliser et celle que j'utiliserai dans le TP.

1. La carte Arduino Nano

La carte Arduino Nano (Figure 1) est basée sur un ATMega328 cadencé à 16 MHz. Sa mémoire de 32 kB et son grand nombre d'E/S font de ce circuit compatible DIL30 un élément idéal pour les *systèmes embarqués* ou pour des applications robotiques nécessitant du multitâches.

La Nano 3.0 peut se programmer avec le logiciel Arduino. Le contrôleur ATMega328 contient un bootloader qui permet de modifier le programme sans passer par un programmateur.

Figure. 1 La carte Arduino Nano.

2. La carte Arduino Uno

2.1. Présentation

La carte Arduino Uno diffère de toutes les cartes car elle n'utilise pas le circuit intégré FTDI usb-vers-série. A la place, elle utilise un Atmega8U2 programmé en convertisseur USB-verssérie.

"Uno" signifie un(1) en Italien et ce nom marque la venue prochaine de la version 1.0 du logiciel Arduino. La carte UNO et la version 1.0 du logiciel seront la référence des versions Arduino à venir. La carte Uno est la dernière d'une série de carte USB Arduino, et le modèle de référence des plateformes Arduino; pour une comparaison avec les versions précédentes, voir l'index des cartes Arduino.

Elle contient tout ce qui est nécessaire pour le fonctionnement du microcontrôleur; Pour pouvoir l'utiliser et se lancer, il suffit simplement de la connecter à un ordinateur à l'aide d'un câble USB (ou de l'alimenter avec un adaptateur secteur ou une pile, mais ceci n'est pas indispensable, l'alimentation étant fournie par le port USB).

2.2. Brochage de la carte Uno

La carte Arduino Uno est une carte à microcontrôleur basée sur l'ATmega328. Elle dispose (voir figure. 2):

1. Un microcontrôleur,
2. d'une connexion USB,
3. d'un connecteur d'alimentation jack,
4. LED de visualisation
5. a) de 14 broches numériques d'entrées/sorties (dont 6 peuvent être utilisées en sorties PWM (largeur d'impulsion modulée)),
5. b) de 6 entrées analogiques (qui peuvent également être utilisées en broches entrées/sorties numériques),
6. d'un quartz 16Mhz,
7. d'un connecteur ICSP (programmation "in-circuit"),
8. un bouton de réinitialisation (reset).

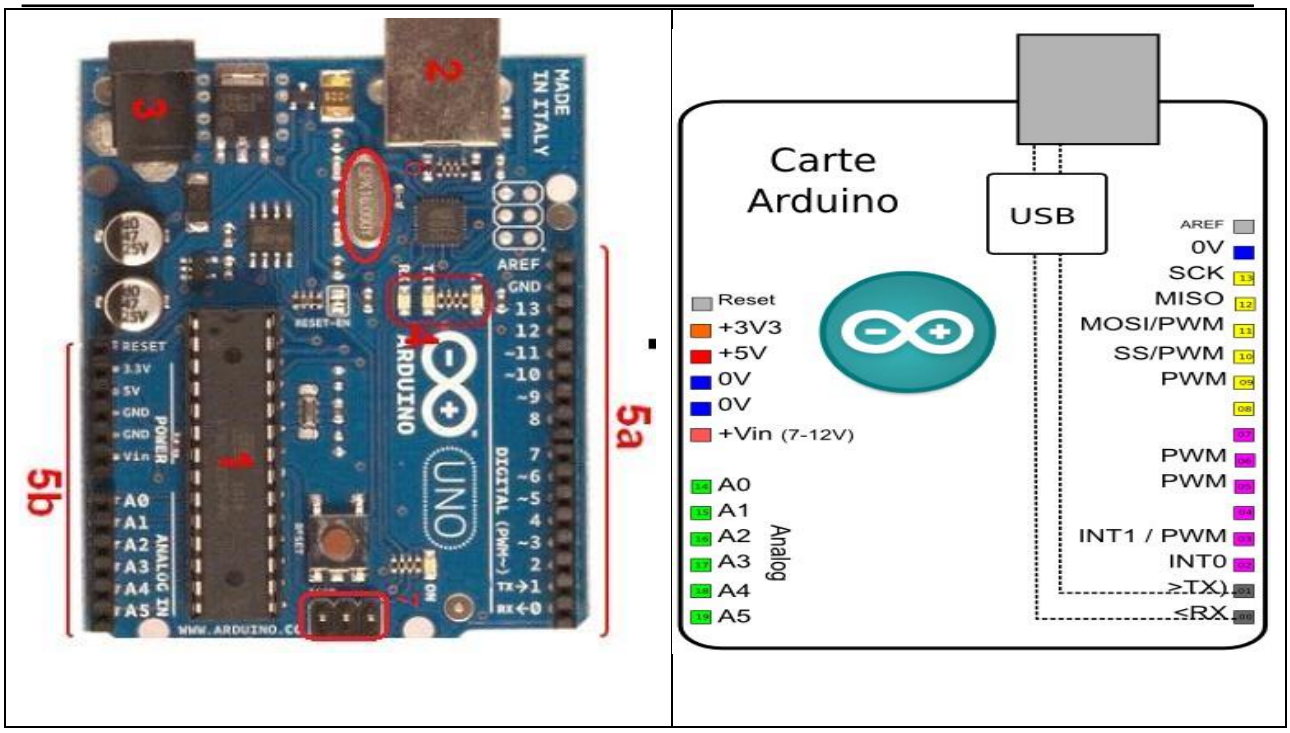


Figure. 2 La carte Arduino UNO

Alimentation

La carte Arduino Uno peut-être alimentée soit via la connexion USB (qui fournit 5V jusqu'à 500mA) ou à l'aide d'une alimentation externe. La source d'alimentation est sélectionnée automatiquement par la carte.

L'alimentation externe (non-USB) peut être soit un adapteur secteur (pouvant fournir typiquement de 3V à 12V sous 500mA) ou des piles (ou des accus). L'adaptateur secteur peut être connecté en branchant une prise 2.1mm positif au centre dans le connecteur jack de la carte. Les fils en provenance d'un bloc de piles ou d'accus peuvent être insérés dans les connecteurs des broches de la carte appelées Gnd (masse ou 0V) et Vin (Tension positive en entrée) du connecteur d'alimentation.

La carte peut fonctionner avec une alimentation externe de 6 à 20 volts. Cependant, si la carte est alimentée avec moins de 7V, la broche 5V pourrait fournir moins de 5V et la carte pourrait être instable. Si on utilise plus de 12V, le régulateur de tension de la carte pourrait chauffer et endommager la carte. Aussi, la plage idéale recommandée pour alimenter la carte Uno est entre 7V et 12V.

Les broches d'alimentation sont les suivantes :

- VIN. La tension d'entrée positive lorsque la carte Arduino est utilisée avec une source de tension externe (à distinguer du 5V de la connexion USB ou autre source 5V régulée). Vous

pouvez alimenter la carte à l'aide de cette broche, ou, si l'alimentation est fournie par le jack d'alimentation, accéder à la tension d'alimentation sur cette broche.

- 5V. La tension régulée utilisée pour faire fonctionner le microcontrôleur et les autres composants de la carte (pour info : les circuits électroniques numériques nécessitent une tension d'alimentation parfaitement stable dite "tension régulée" obtenue à l'aide d'un composant appelé un régulateur et qui est intégré à la carte Arduino). Le 5V régulé fourni par cette broche peut donc provenir soit de la tension d'alimentation VIN via le régulateur de la carte, ou bien de la connexion USB (qui fournit du 5V régulé) ou de tout autre source d'alimentation régulée.
- 3V3. Une alimentation de 3.3V fournie par le circuit intégré FTDI (circuit intégré faisant l'adaptation du signal entre le port USB de votre ordinateur et le port série de l'ATmega) de la carte est disponible : ceci est intéressant pour certains circuits externes nécessitant cette tension au lieu du 5V). L'intensité maximale disponible sur cette broche est de 50mA GND. Broche de masse (ou 0V).

Mémoire

L'ATmega 328 a 32Ko de mémoire FLASH pour stocker le programme (dont 0.5Ko également utilisés par le bootloader). L'ATmega 328 a également 2ko de mémoire SRAM (volatile) et 1Ko d'EEPROM.

Pour info : Le bootloader est un programme préprogrammé une fois pour toute dans l'ATméga et qui permet la communication entre l'ATmega et le logiciel Arduino via le port USB, notamment lors de chaque programmation de la carte.

Entrées et sorties numériques

Chacune des 14 broches numériques de la carte UNO (numérotées des 0 à 13) peut être utilisée soit comme une entrée numérique, soit comme une sortie numérique, en utilisant les instructions pinMode(), digitalWrite() et digitalRead() du langage Arduino. Ces broches fonctionnent en 5V. Chaque broche peut fournir ou recevoir un maximum de 40mA d'intensité et dispose d'une résistance interne de "rappel au plus" (pull-up) (déconnectée par défaut) de 2050 KOhms. Cette résistance interne s'active sur une broche en entrée à l'aide de l'instruction digitalWrite(broche, HIGH).

De plus, certaines broches ont des fonctions spécialisées :

- **Communication Serie**: Broches 0 (RX) et 1 (TX). Utilisées pour recevoir (RX) et transmettre (TX) les données séries de niveau TTL. Ces broches sont connectées aux broches correspondantes du circuit intégré ATmega8U2 programmé en convertisseur USBvers-série

de la carte, composant qui assure l'interface entre les niveaux TTL et le port USB de l'ordinateur.

- **Interruptions Externes**: Broches 2 et 3. Ces broches peuvent être configurées pour déclencher une interruption sur une valeur basse, sur un front montant ou descendant, ou sur un changement de valeur. Voir l'instruction attachInterrupt() pour plus de détails.
- **Impulsion PWM (largeur d'impulsion modulée)**: Broches 3, 5, 6, 9, 10, et 11. Fournissent une impulsion PWM 8-bits à l'aide de l'instruction analogWrite().
- **SPI (Interface Série Périphérique)**: Broches 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Ces broches supportent la communication SPI (Interface Série Périphérique) disponible avec la librairie pour communication SPI. Les broches SPI sont également connectées sur le connecteur ICSP qui est mécaniquement compatible avec les cartes Mega.
- **I2C**: Broches 4 (SDA) et 5 (SCL). Supportent les communications de protocole I2C (ou interface TWI (Two Wire Interface - Interface "2 fils"), disponible en utilisant la librairie Wire/I2C (ou TWI - Two-Wire interface - interface "2 fils") .
- **LED**: Broche 13. Il y a une LED incluse dans la carte connectée à la broche 13. Lorsque la broche est au niveau HAUT, la LED est allumée, lorsque la broche est au niveau BAS, la LED est éteinte.

Broches analogiques

La carte Uno dispose de 6 entrées analogiques (numérotées de 0 à 5), chacune pouvant fournir une mesure d'une résolution de 10 bits (càd sur 1024 niveaux soit de 0 à 1023) à l'aide de la très utile fonction *analogRead()* du langage Arduino. Par défaut, ces broches mesurent entre le 0V (valeur 0) et le 5V (valeur 1023), mais il est possible de modifier la référence supérieure de la plage de mesure en utilisant la broche AREF et l'instruction *analogReference()* du langage Arduino.

Note : les broches analogiques peuvent être utilisées en tant que broches numériques : elles sont numérotées en tant que broches numériques de 14 à 19.

Autres broches

Il y a deux autres broches disponibles sur la carte :

- **AREF** : Tension de référence pour les entrées analogiques (si différent du 5V). Utilisée avec l'instruction *analogReference()*.
- **Reset** : Mettre cette broche au niveau BAS entraîne la réinitialisation (= le redémarrage) du microcontrôleur. Typiquement, cette broche est utilisée pour ajouter un bouton de réinitialisation sur le circuit qui bloque celui présent sur la carte.

Communication

La carte Arduino Uno dispose de toute une série de facilités pour communiquer avec un ordinateur, une autre carte Arduino, ou avec d'autres microcontrôleurs. L'ATmega 328 dispose d'une UART (Universal Asynchronous Receiver Transmitter ou émetteur-récepteur asynchrone universel en français) pour communication série de niveau TTL (5V) et qui est disponible sur les broches 0 (RX) et 1 (TX). Un circuit intégré ATmega8U2 sur la carte assure la connexion entre cette communication série vers le port USB de l'ordinateur et apparaît comme un port COM virtuel pour les logiciels de l'ordinateur. Le code utilisé pour programmer l'ATmega8U2 utilise le driver standard USB COM, et aucun autre driver externe n'est nécessaire. Cependant, sous Windows, un fichier .inf est requis.

Le logiciel Arduino inclut une fenêtre terminal série (ou moniteur série) sur l'ordinateur et qui permet d'envoyer des textes simples depuis et vers la carte Arduino. Les LEDs RX et TX sur la carte clignote lorsque les données sont transmises via le circuit intégré USB-vers-série et la connexion USB vers l'ordinateur (mais pas pour les communications série sur les broches 0 et 1).

Une librairie Série Logicielle permet également la communication série (limitée cependant) sur n'importe quelle broche numérique de la carte UNO.

L'ATmega 328 supporte également la communication par protocole I2C (ou interface TWI

(Two Wire Interface - Interface "2 fils") et SPI.

2.3. Synthèse des caractéristiques

Microcontrôleur	ATmega328
Tension de fonctionnement	5V
Tension d'alimentation (recommandée)	7-12V
Tension d'alimentation (limites)	6-20V
Broches E/S numériques	14 (dont 6 disposent d'une sortie PWM)
Broches d'entrées analogiques	6 (utilisables en broches E/S numériques)
Intensité maxi disponible par broche E/S (5V)	40 mA (*ATTENTION : 200mA cumulé pour l'ensemble des broches E/S)*
Intensité maxi disponible pour la sortie 3.3V	50 mA
Intensité maxi disponible pour la sortie 5V	Fonction de l'alimentation utilisée - 500 mA
	max si port USB utilisé seul

Mémoire Programme Flash	32 KB (ATmega328) dont 0.5 KB sont utilisés par le bootloader
Mémoire SRAM (mémoire volatile)	2 KB (ATmega328)
Mémoire EEPROM (mémoire non volatile)	1 KB (ATmega328)
Vitesse d'horloge	16 Hz

3. La carte Arduino Zero

La carte Arduino Zero (Figure 3) est une simple et puissante 32-bit extension de la plate-forme établie par l'ONU. Le conseil Zéro élargit la famille en fournissant des performances accrues, permettant une variété de possibilités de projets pour les appareils, et agit comme un outil éducatif grand pour apprendre sur le développement d'applications 32 bits.

Les applications Zero s'étendent des dispositifs intelligents d'IoT, de la technologie wearable, de l'automatisation high-tech, à la robotique fou. La carte est alimentée par le SAMD21 MCU d'Atmel, qui comporte un noyau ARM Cortex® M0 + de 32 bits. L'une de ses caractéristiques les plus importantes est le débogueur embarqué Atmel (EDBG), qui fournit une interface de débogage complète sans avoir besoin de matériel supplémentaire, ce qui augmente considérablement la facilité d'utilisation pour le débogage logiciel. EDBG prend également en charge un port COM virtuel qui peut être utilisé pour la programmation de périphériques et de boot loaders.

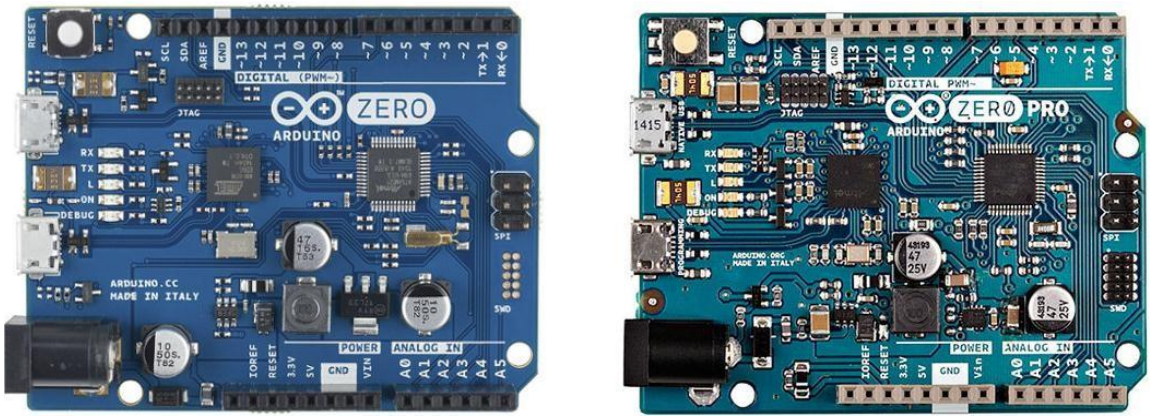


Figure. 3 La carte Arduino Zero.

4. La carte Arduino Mega

La carte Arduino Mega (Figure 4) est une autre carte qui offre toutes les fonctionnalités des précédentes, mais avec des options en plus. On retrouve notamment un nombre d'entrées et de sorties plus importantes ainsi que plusieurs liaisons séries.

Figure. 4 La carte Arduino "Mega"

5. La carte Arduino Leonardo

Est une carte microcontrôleur basée sur l'ATmega32u4. Il dispose de 20 broches numériques d'entrée/sortie (dont 7 peuvent être utilisées comme sorties PWM et 12 entrées en tant analogiques), un oscillateur à quartz 16 MHz, une connexion micro USB, une prise d'alimentation, d'une embase ICSP et un bouton de réinitialisation. Il contient tout le nécessaire pour soutenir le microcontrôleur; simplement le connecter à un ordinateur avec un câble USB ou de la puissance avec un adaptateur ou d'une batterie AC-DC pour commencer. Le Leonardo diffère de toutes les cartes précédentes en ce que le ATmega32u4 a intégré la communication USB, ce qui élimine la nécessité d'un processeur secondaire. Cela permet à la Leonardo apparaisse à un ordinateur connecté comme une souris et un clavier, en plus d'un (CDC) port série / COM virtuel.

Figure. 5 La carte Arduino « Leonardo »

V. Circuits additionnels

Il est possible de spécialiser la carte Arduino en l'associant avec des circuits additionnels que l'on peut fabriquer soi-même ou acheter déjà montés. Lorsqu'ils se branchent directement sur la carte, ces circuits s'appellent des « sheds » ou cartes d'extension. Ces circuits spécialisés apportent au système des fonctionnalités diverses et étendues dont voici quelques exemples :

- Ethernet : communication réseau ;
- Bluetooth ou zigbee : communication sans fil ;
- Pilotage de moteurs (pas à pas ou à courant continu) ;
- Pilotage de matrices de LEDs : pour piloter de nombreuses LEDs avec peu de sorties ;
- Ecran LCD : pour afficher des informations ;
- Lecteur de carte mémoire : lire ou stocker des données ;
- Lecteur de MP3 ;
- GPS : pour avoir une information de position géographique ; • Joystick ;

Outil de Développement IDE d'Arduino

I. Introduction

Le logiciel qui permet de programmer votre carte Arduino porte le nom d'IDE, ce qui signifie Integrated Development Environment ou encore Environnement de Développement Intégré.
En effet, cette application intègre l'édition des programmes, le téléversement dans la carte Arduino et plusieurs bibliothèques. L'IDE existe pour les trois systèmes d'exploitation et cet article vous explique comment l'installer dans votre ordinateur.

II. Installation

L'installation de cet outil dépend du système d'exploitation de votre ordinateur. Nous allons donc détailler les différentes étapes pour le système Windows. La figure 1 montre la partie de l'onglet « Software » du site www.arduino.cc ou www.arduino.org permettant de choisir l'IDE en fonction de son système d'exploitation.

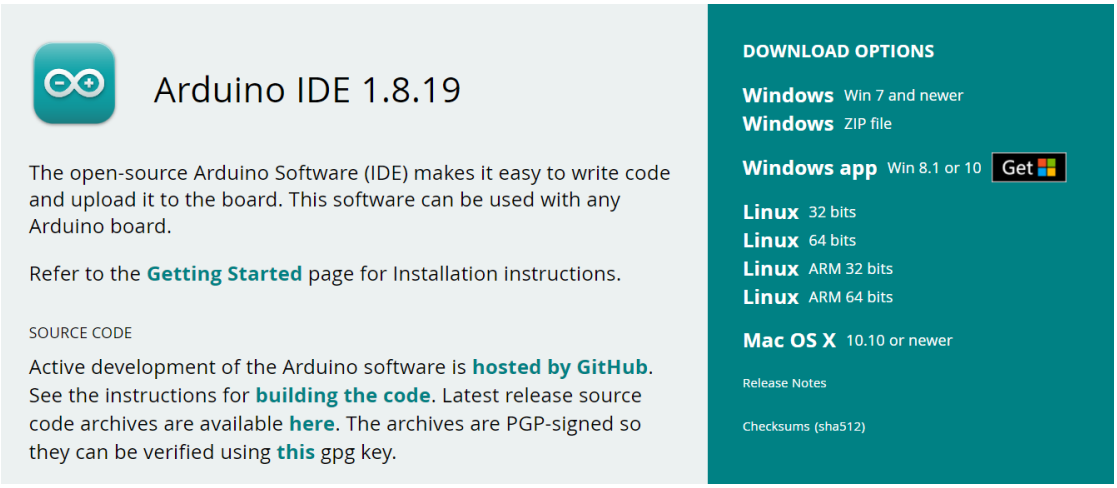


Figure. 6 Extraite du site arduino.cc

Le logiciel convient de Windows XP à Windows 10. Si c'est la première installation de l'IDE sur votre ordinateur, nous vous conseillons de choisir dans l'onglet « Software » l'option « Windows Installer » qui est une version installable ; il suffit alors de suivre les différentes étapes et les pilotes (drivers) pour les cartes Arduino seront également installés.
Lancez l'IDE fraîchement installé. Dans le menu Fichier, allez chercher le programme Blink donné en exemple (Fichier -> Exemples -> 01. Basic -> Blink). Connectez votre carte Arduino en USB à votre ordinateur. Si votre carte est une carte Uno, vous devez lire en bas à droite de

l'IDE « Arduino/Nano ATmega328 sur COM7 » (Nano et 7 sont des exemples). Avant de téléverser ce programme dans votre carte Nano, vérifiez dans **Outils** que le type de carte est bien une Nano et que le port est bien COM7 (Figure 7), sinon changer les options. Téléversez le programme et observez la LED qui clignote à la fréquence de 2 Hz (une seconde allumée et une seconde éteinte). Si c'est le cas, l'IDE est parfaitement installé et vous allez pouvoir programmer vos cartes.

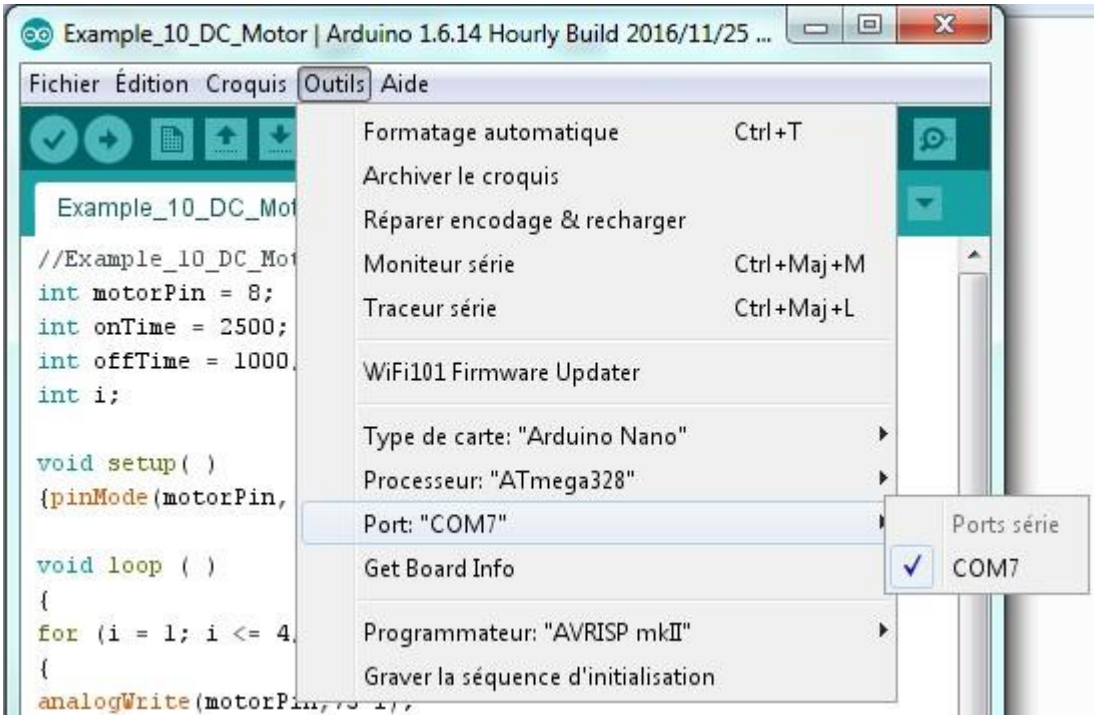


Figure. 7 Connexion de la carte Arduino

III. Présentation de l'interface IDE 1. Description

Le logiciel IDE Arduino a pour fonctions principales :

- De pouvoir écrire et compiler des programmes pour la carte Arduino
- De se connecter avec la carte Arduino pour y transférer les programmes
- De communiquer avec la carte Arduino

Cet environnement de développement intégré (EDI) dédié au langage Arduino et à la programmation des cartes Arduino comporte (Figure. 8) :

- Une **BARRE DE MENUS** comme pour tout logiciel une interface graphique (GUI),
- Une **BARRE DE BOUTONS** qui donne un accès direct aux fonctions essentielles du logiciel et fait toute sa simplicité d'utilisation,
- Un **EDITEUR** (à coloration syntaxique) pour écrire le code de vos programmes, avec onglets de navigation,

- Une **ZONE DE MESSAGES** qui affiche indique l'état des actions en cours,
- Une **CONSOLE TEXTE** qui affiche les messages concernant le résultat de la compilation du programme.

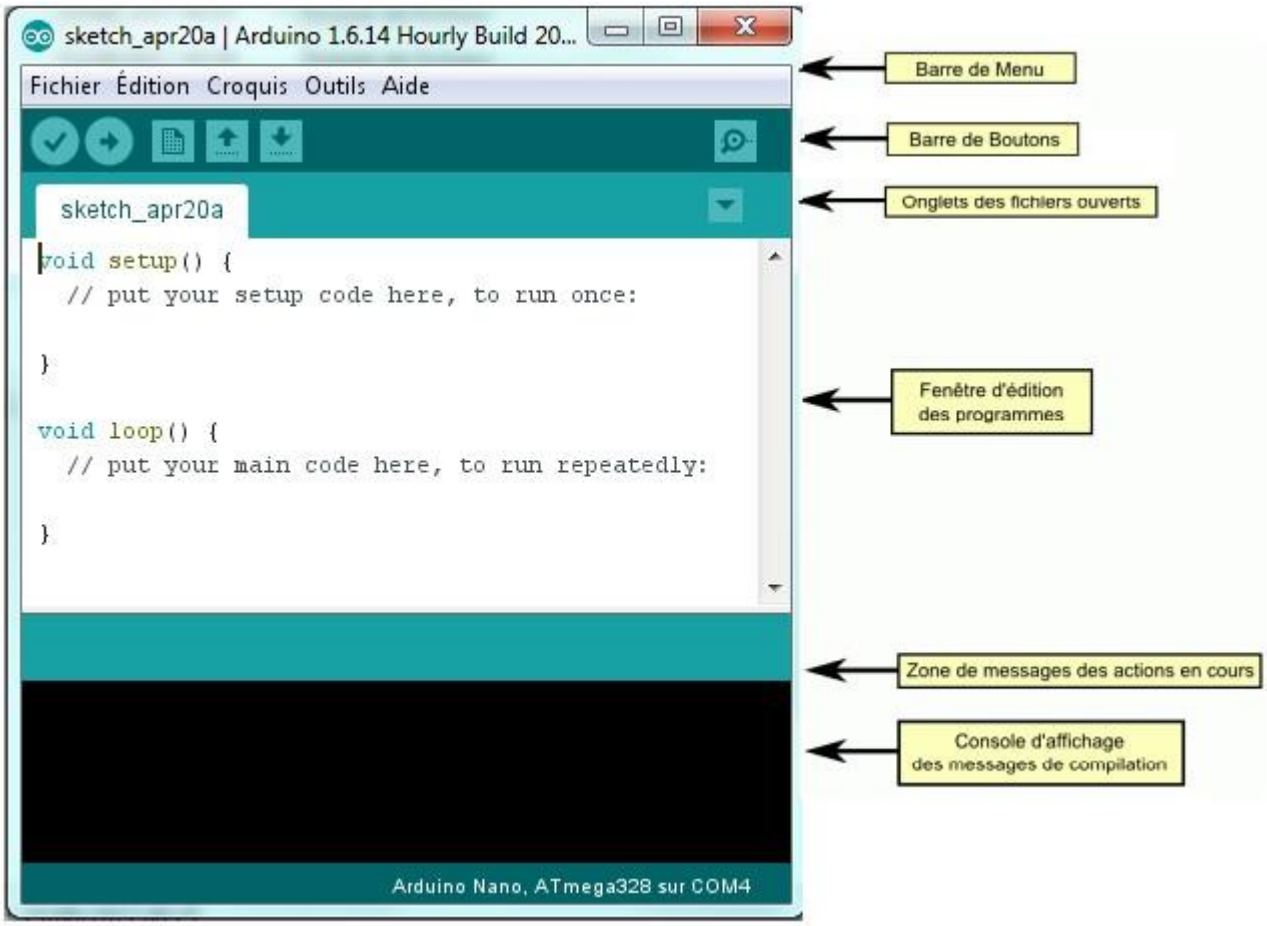


Figure. 8 Présentation de la fenêtre IDE Arduino

Le logiciel Arduino intègre également :

- Un **TERMINAL SERIE** (Figure. 9) qui permet d'afficher des messages textes reçus de la carte Arduino et d'envoyer des caractères vers la carte Arduino. Cette fonctionnalité permet une mise au point facilitée des programmes, permettant d'afficher sur l'ordinateur l'état de variables, de résultats de calculs ou de conversions analogique-numérique : un élément essentiel pour améliorer, tester et corriger ses programmes.
- Sur votre ordinateur, il faut ouvrir la fenêtre terminal de l'IDE Arduino : pour ce faire, un simple clic sur le bouton « Sérial Monitor ».

- La fenêtre « Terminal » s'ouvre alors :
- Il faut alors régler le débit de communication sur la même valeur que celle utilisée par le programme avec lequel nous allons programmer la carte Arduino.

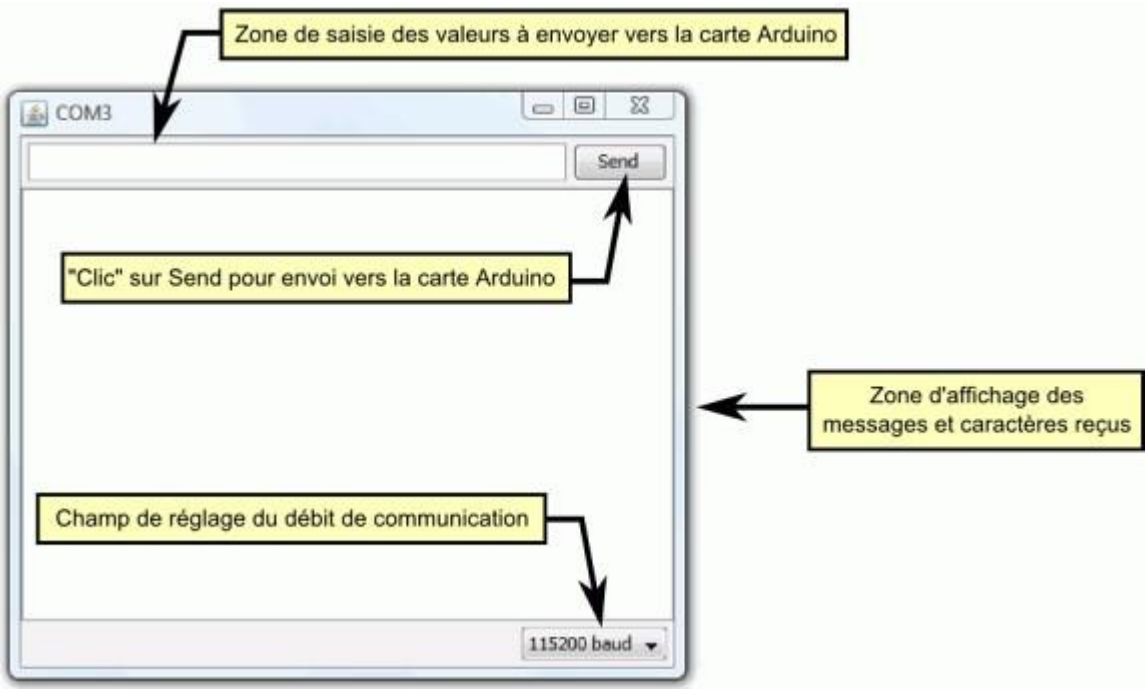


Figure. 9 Le moniteur série (Terminal série).

2. Principe général d'utilisation

Le code écrit avec le logiciel Arduino est appelé un programme (ou une séquence - sketch en anglais) :

- Ces programmes sont écrits dans **l'éditeur de texte**. Celui-ci a les fonctionnalités usuelles de copier/coller et de rechercher/remplacer le texte.
- la **zone de messages** donne l'état de l'opération en cours lors des sauvegardes, des exportations et affiche également les erreurs.
- La **console texte** affiche les messages produits par le logiciel Arduino incluant des messages d'erreur détaillés et autres informations utiles.
- la **barre de boutons** vous permet de vérifier la syntaxe et de transférer les programmes, créer, ouvrir et sauver votre code, et ouvrir le moniteur série.
- la **barre des menus** vous permet d'accéder à toutes les fonctionnalités du logiciel Arduino.

2.1. Description de la barre des boutons

La barre des boutons comporte (voir les détaille figure. 10) :

Vérifier/compiler : Vérifie le code à la recherche d'erreur.

Stop : Stoppe le moniteur série ou les autres boutons activés.

Nouveau : Crée un nouveau code (ouvre une fenêtre d'édition vide).

Ouvrir : Ouvre la liste de tous les programmes dans votre "livre de programmes". Cliquer sur l'un des programmes l'ouvre dans la fenêtre courante.

Note: en raison d'un bug dans Java, ce menu ne défile pas. Si vous avez besoin d'ouvrir un programme loin dans la liste, utiliser plutôt le menu File > Sketchbook.

Sauver : Enregistre votre programme.

Transférer vers la carte : Compile votre code et le transféré vers la carte Arduino. Voir cidessous "Transférer les programmes" pour les détails.

Moniteur Série : Ouvre la fenêtre du moniteur (ou terminal) série.

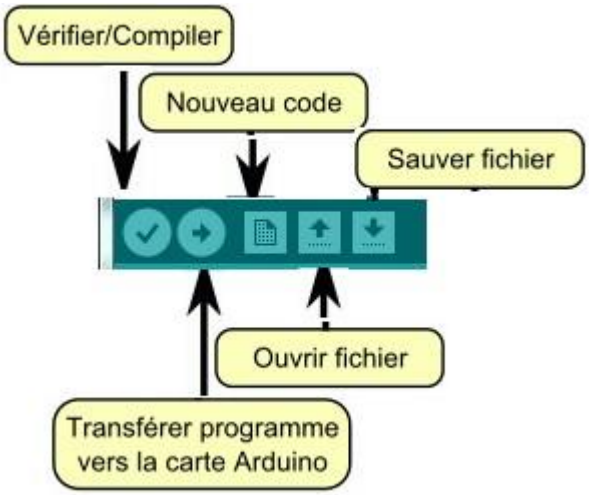


Figure. 10 Détaille de la barre des Boutons.

2.2. Transfert des programmes vers la carte Arduino

- Pour première programmation de la carte, aller dans le menu **File>Examples>Digital>Blink**: un programme s'ouvre dans la fenêtre éditeur.
- Appuyez alors sur le bouton **Vérifier** de la barre d'outils pour lancer la vérification du code.
- Si tout va bien, aucun message d'erreur ne doit apparaître dans la console et la zone de message doit afficher **Compilation terminée** attestant que la vérification s'est bien déroulée comme montre la figure. 11.

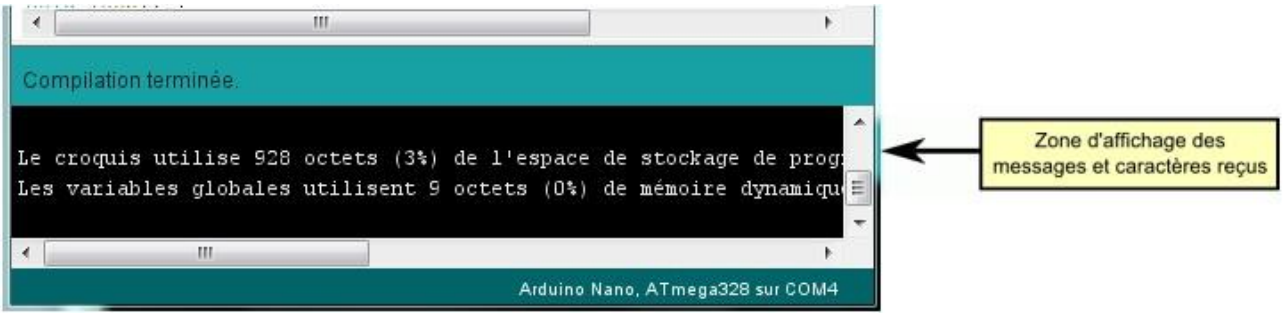


Figure 11. La console IDE Arduino.

Avant de transférer votre programme vers la carte Arduino, vous devez vérifier que vous avez bien sélectionné la bonne carte Arduino depuis le menu **Tools>Board** (Outils>Carte). Les cartes sont décrites ci-dessous. Votre carte doit évidemment être connectée à l'ordinateur via un câble USB.

Vous devez également sélectionner le bon port série depuis le menu **Tools > Serial Port** (Outils > Port Série) :

- Sur un Mac, le port série ressemble probablement à quelque chose comme /dev/tty.usbserial-1B1 (pour une carte USB), ou /dev/tty.USA19QW1b1P1.1 (pour une carte série connectée avec un adaptateur USB-vers-Série).
- Sous Winsows, c'est probablement COM1 ou COM2 (pour une carte série) ou COM4, COM5, COM7 ou supérieur (pour une carte USB) - pour trouver le bon, vous pouvez chercher dans la rubrique des ports série USB dans la section des ports du panneau de configuration ou du gestionnaire de périphériques.
- Sous Linux, çà devrait être /dev/ttyUSB0, /dev/ttyUSB1 ou équivalent.

Une fois que vous avez sélectionné le bon port série et la bonne carte Arduino, cliquez sur le bouton **Téléverser** (Transfert vers la carte) dans la barre d'outils (Figure 12), ou bien sélectionner le menu **Croquis>Téléverser (ctrl+U)**. Avec les versions récentes (Duemilanove notamment), la carte Arduino va alors automatiquement se réinitialiser et démarrer le transfert. Avec les versions précédentes qui ne sont pas équipées de l'auto-réinitialisation, vous devez appuyer sur le bouton "reset" de la carte juste avant de démarrer le transfert.

Figure 12. La téléversement du programme vers la carte Arduino.

LE Langage Arduino

➢ Structure globale d'un programme langage Arduino

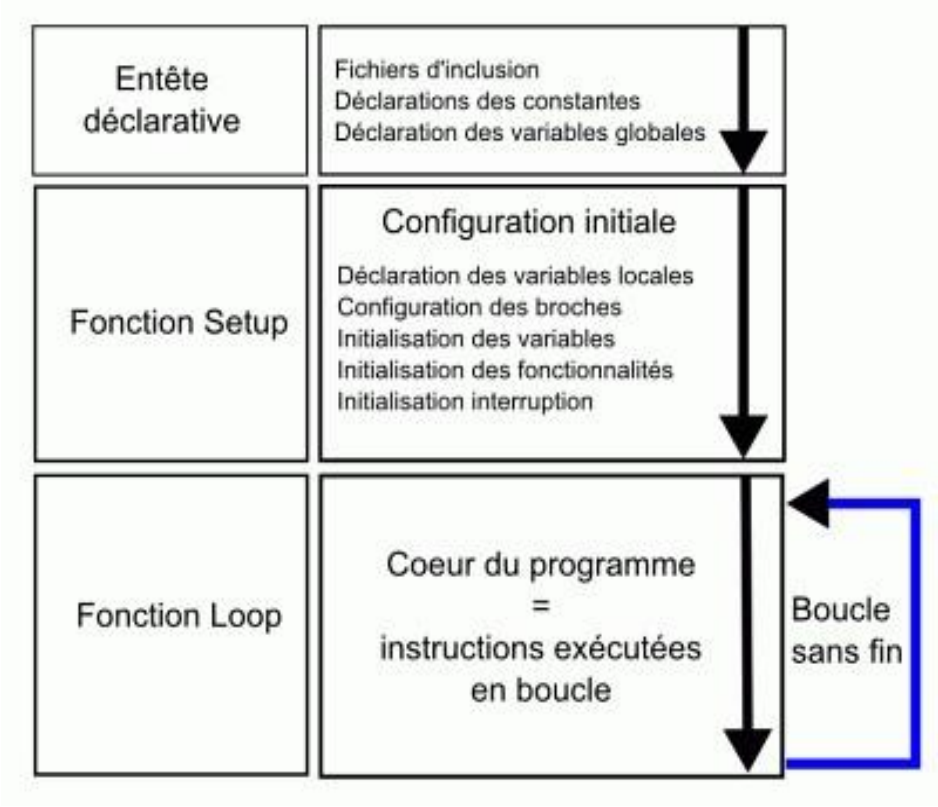
Le programme Arduino comprend :

- Une entête déclarative
- Une partie « configuration » qui ne sera exécutée qu'une fois (fonction setup( ))
- Une partie constituée d'une boucle sans fin que le programme répètera à l'infini « fonction loop( ) » : c'est le cœur du programme.

➢ Déroulement du programme

Le programme se déroule de la façon suivante :

- Après avoir pris en compte les instructions de la partie déclarative,
- puis après avoir exécuté la partie configuration (fonction setup( )),
- le programme bouclera sans fin (fonction loop ( )), exécutant de façon répétée le code compris dans la boucle sans fin.



Le déroulement d'un programme Langage Arduino

Les programmes Arduino peuvent être divisés en trois parties principales : la structure, les valeurs (variables et constantes) et les fonctions. Le langage Arduino est basé sur les langages C/C++.

I- Structure

1/ Fonctions de base :

Ces deux fonctions sont obligatoires dans tout programme en langage Arduino :

- void setup()
- void loop()

Description setup()

La fonction setup() est appelée au démarrage du programme. Cette fonction est utilisée pour initialiser les variables, le sens des broches, les librairies utilisées. La fonction setup n'est exécutée qu'une seule fois, après chaque mise sous tension ou reset (réinitialisation) de la carte Arduino.

Syntaxe :

```
void setup()
{
…
}
```

Description : loop ()

Après avoir créé une fonction **setup()**, qui initialise et fixe les valeurs de démarrage du programme, la fonction **loop ()** (boucle en anglais) fait exactement ce que son nom suggère et s'exécute en boucle sans fin, permettant à votre programme de s'exécuter et de répondre. Utiliser cette fonction pour contrôler activement la carte Arduino. **Syntaxe :**

```
void loop () {
….
}
```

Exemple :

```
int buttonPin = 3; // déclare une variable entière nommée buttonPin et  mémorisé à une broche utilisée avec un bouton poussoir
void setup()    // la fonction setup initialise la communication série et une broche utilisée avec un bouton poussoir

{
 Serial.begin(9600);
 pinMode(buttonPin, INPUT); }
// la fonction loop teste l'état du bouton à chaque passage et envoie au PC une lettre H si il est appuyé, L sinon.
void loop()
{
 if (digitalRead(buttonPin) == HIGH)
   Serial.write('H');
else
   Serial.write('L');
```

```
delay(1000);
}
```

2/ Structures de contrôle :

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return

Description: Le test IF … ELSE IF…ELSE

L'instruction if ("si" en français), utilisée avec un opérateur logique de comparaison, permet de tester si une condition est vraie, par exemple si la mesure d'une entrée analogique est bien supérieure à une certaine valeur.

Syntaxe :

Le format d'un test **if …else if … else** est le suivant :

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

Description : Boucle FOR

L'instruction **for** est utilisée pour répéter l'exécution d'un bloc d'instructions regroupées entre des accolades. Un compteur incrémental est habituellement utilisé pour incrémenter et finir la boucle. L'instruction **for** est très utile pour toutes les opérations répétitives et est souvent utilisées en association avec des tableaux de variables pour agir sur un ensemble de données ou broches.

Il y a 3 parties dans l'entête d'une boucle for :

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

L'initialisation a lieu en premier et une seule fois. A chaque exécution de la boucle, la condition est testée; si elle est VRAIE, le bloc d'instructions et l'incrementation sont exécutés. puis la condition est testée de nouveau. Lorsque la condition devient FAUSSE, la boucle stoppe.

Exemple :

```
// Eteint progressivement une LED en utilisant une broche PWM (impulsion)

int PWMpin = 10; // LED en série avec une résistance de 1k sur la broche 10

void setup()
{
  // aucune initialisation nécessaire
}
void loop()
{
   // boucle incrémentant la variable i de 0 à 255, de 1 en 1
for (int i=0; i <= 255; i++){

     // impulsion de ratio HAUT/BAS fonction de i sur la broche 10
analogWrite(PWMpin, i);

     delay(10); // pause de 10ms

   } // fin de la boucle for
}
```

Description : switch / case

Tout comme l'instruction **if**, l'instruction **switch / case** ("commutateur de cas" en anglais) contrôle le déroulement des programmes. L'instruction **switch / case** permet au programmeur de construire une liste de "cas" (ou possibilités) à l'intérieur d'accolades. Le programme teste chaque cas avec la variable de test, et exécute le code si une correspondance (un test VRAI) est trouvé.

switch / case est légèrement plus flexible qu'une structure **if / else** en ce sens que le programmeur peut définir si la structure **switch** devra continuer à tester les cas sous la forme d'une liste, même après avoir trouvé une correspondance. Si l'instruction **break** n'est pas trouvée après avoir exécuté le code d'une condition vraie, le programme continuera à tester les conditions restantes parmi les cas restants. Si une instruction **break** est rencontrée, le cas fait sortir de la structure, de la même façon que pour une construction **if / else if**.

Paramètres

var : variable dont vous vouler tester l'état **default** : si aucune autre condition vraie n'est rencontrée, le code default sera exécuté **break** : Sans une instruction break, l'instruction switch continuera à exécuter les expressions suivantes ("en tombant dedans") jusqu'à un rencontrer un break ou jusqu'à la fin du switch. Si une est trouvée, le code sera exécuté, même si ce n'est pas votre intention. L'instruction

break indique à l'instruction **switch** d'arrêter de rechercher des conditions vraies, et fait sortir de la structure de choix.

Exemple / Syntaxe :

```
switch (var) { // debut de la structure
case 1: // cas 1
     //faire quelque chose quand la variable est égale à 1 (càd sir var == 1)
break;
     // l'instruction break est en option
   case 2: // cas 2
      //faire quelque chose quand la variable est égale à 2 (càd sir var == 2)
break;
     // l'instruction break est en option

   default: // cas par défaut
     // si aucune condition n'est vraie, le code par défaut sera
exécuté      // le cas default est optionnel (non -obligatoire)   }
```

Description: Boucle While

Les boucles **while** ("tant que" en anglais) bouclent sans fin, et indéfiniment, jusqu'à ce que la condition ou l'expression entre les parenthèses ( ) devienne fausse. Quelque chose doit modifier la variable testée, sinon la boucle **while** ne se terminera jamais. Cela peut être dans votre code, soit une variable incrémentée, ou également une condition externe, soit le test d'un capteur.

Syntaxe :

```
while(expression){ // tant que l'expression est vraie

 // instructions à effectuer

}
```

Paramètres :

expression : Une instruction (booléenne) C qui renvoie un résultat VRAI ou FAUX.

Exemple :

```
var = 0;
while(var < 200){  // tant que la variable est inférieur à 200

 // fait quelque chose 200 fois de suite...

 var++; // incrémente la variable

}
```

Description : Boucle Do/ while

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. **Syntaxe :**

```
do // faire... {
   // bloc d'instruction

} while (condition); // tant que la condition est vraie
```

Paramètres :

condition : expression booléenne dont le résultat peut être VRAI ou FAUX. **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
```

3/ Syntaxe de base

- ; (point virgule)
- {} (accolades)
- // (commentaire sur une ligne)
- /* */ (commentaire sur plusieurs lignes)
- #define
- #include

4/Opérateurs arithmétiques

- = (égalité)
- + (addition)
- - (soustraction)
- * (multiplication)
- / (division)

5/ Opérateurs de comparaison

- == (égal à)
- != (différent de)
- < (inférieur à)
- > (supérieur à)
- <= (inférieur ou égal à)
- >= (supérieur ou égal à)

6/ Opérateurs booléens

- && (ET booléen)
- || (OU booléen)
- ! (NON booléen)

7/ Opérateurs composés

- ++ (incrémentation)

- -- (décrémentation) (à revoir)
- += (addition composée)
- -= (soustraction composée)
- *= (multiplication composée)
- /= (division composée) **Description : #define**

L'instruction **#define** est un élément très utile du langage C qui permet au programmeur de donner un nom à une constante avant que le programme soit compilé (pour info : en C, les instructions précédés du # sont des instructions utilisées par le pré-compilateur). Les constantes ainsi définie dans le langage Arduino ne prennent aucune place supplémentaire en mémoire dans le microcontrôleur. Le compilateur remplacera les références à ces constantes par la valeur définie au moment de la compilation.
Ceci peut cependant avoir quelques effets indésirables, si par exemple, un nom de constante qui a été défini par #define est inclus dans d'autres constantes ou nom de variable. Dans ce cas, le texte de ces constantes ou de ces variables sera remplacé par la valeur définie avec #define.
D'une manière générale, le mot clé const est préférable pour définir les constantes et doit être utilisé plutôt que #define.

Syntaxe :

L'instruction Arduino #define a la même syntaxe que le #define du langage C :

```
#define constantName   value
```

Noter que le signe # est nécessaire. En langage C, les instructions précédées par # sont des instructions utilisées par le pré-compilateur.

Exemple :

```
#define ledPin 3
// Le compilateur remplacera tout texte ledPin avec la valeur 3 au moment de la compilation
```

Conseils

- Il n'y a pas de point-virgule après l'instruction #define (il en est de même pour toutes les instructions de pré-compilation débutant par #). Si vous en incluez un, le compilateur vous donnera des messages d'erreur énigmatiques dans la console du logiciel Arduino lors de la compilation.

```
#define  ledPin 3;    // ceci est une erreur - ne pas mettre de ;
```

- De la même façon, inclure un signe égal après l'instruction #define vous donnera des messages d'erreur énigmatiques dans la console du logiciel Arduino lors de la compilation.

```
#define ledPin  = 3  // ceci est également une erreur ne pas mettre de =
```

Description : #include

L'instruction **#include** est utilisée pour insérer des librairies externes dans votre programme. Ceci donne accès au programmeur à une large groupe de librairies standard en langage C (groupes de fonctions pré-fabriquées), et également bien sûr des librairies écrites spécialement pour Arduino.

La page de la référence principale pour les librairies AVR en langage C (AVR est la référence des microcontrôleurs Atmel utilisé avec les cartes Arduino) est ici.
Noter que l'instruction #include (qui est une instruction destinée au pré-compilateur), de même qu'avec #define, n'a pas besoin de point-virgule en fin de ligne. Le compilateur donnera un message d'erreur énigmatique si vous en ajoutez une.

Exemple :

Dans cet exemple, on inclut une librairie C qui est utilisée pour mettre les données dans la mémoire Flash programme au lieu de la RAM. Ceci économise de l'espace en Ram pour les besoins en mémoire dynamique et rend la consultation des grands tableaux plus pratique.

```
#include <avr/pgmspace.h>
prog_uint16_t myConstants[] PROGMEM = {0, 21140, 702 , 9128, 0, 25764, 8456,
0,0,0,0,0,0,0,0,29810,8968,29762,29762,4500};
```

Dans cet autre exemple, on inclut une librairie Arduino pour les afficheurs LCD :

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

// initialise la librairie avec les broches utilisées
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
  // initialise le LCD en mode 16 colonnes x 2 lignes
lcd.begin(16, 2);
  // affiche le texte sur le LCD
lcd.print("hello, world!");
}

void loop()
{
}
```

II- Variables et constantes

Les variables sont des expressions que vous pouvez utilisez dans les programmes pour stocker des valeurs numériques.

1/ Constantes prédéfinies

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

Définition des niveaux de broche, HIGH (HAUT) et LOW (BAS)

Lorsqu'on lit ou on écrit sur une broche numérique, seuls deux états distincts sont possible, la broche ne peut prendre/être qu'à deux valeurs : HIGH (HAUT) ou LOW (BAS).

- **HIGH | LOW**

HIGH

La signification de la constante HIGH (en référence à une broche) est quelque chose de différent selon que la broche est définie comme une ENTREE ou comme une SORTIE.

LOW

Cela signifie que la constante LOW a également une signification différente selon que la broche est configurée en ENTREE ou en SORTIE.

Définition des constantes de broches numériques, INPUT and OUTPUT

- **INPUT | OUTPUT**

Les broches numériques peuvent être utilisées soit en mode INPUT (= en entrée), soit en mode OUTPUT (= en sortie). Modifierle mode de fonctionnement d'une broche du mode INPUT (=ENTREE) en mode OUTPUT(=SORTIE) avec l'instruction pinMode() change complètement le comportement électrique de la broche.

Définition de niveaux logiques, true (vrai) et false (faux) (Constantes de type bouléen= binaire)

- **true | false**

Il existe deux constantes utilisées pour représenté le VRAI et le FAUX dans le langage arduino : true et false.

false (= FAUX)

La constante false est la plus facile des deux à définir. false est définie comme le 0 (zéro).

true (=VRAI)

La constante true est souvent défini comme étant le 1, ce qui est correct

2/ Expressions numériques

- **Expressions numériques entières**

Description

Les expressions numériques entières sont des nombres utilisés directement dans un programme, tel que 123. Par défaut, ces nombres sont traités comme des variables de type int mais vous pouvez changer cela avec les "modificateurs" U et L (voir ci-dessous). Normalement, les expressions numériques entières sont traitées comme des entiers en base 10 (base décimale càd notre façon spontanée d'exprimer les nombres), mais des notations spéciales (ou "formateurs") peuvent être utilisé pour saisir des nombres dans d'autres bases.

Base	Exemple	"Formateur"	Commentaire
10 (decimal)	123	aucun	
2 (binaire)	B1111011	B	fonctionne uniquement avec des valeurs 8 bits (0 to 255) et seuls les caractères 0 et 1 sont valides
8 (octal)	0173	0	seuls les caractères de 0-7 sont valides
16 (hexadecimal)	0x7B	0x	les caractères 0-9, A-F et a-f sont valides

Format décimal

Le format decimal correspond à la base 10. C'est les mathématiques du sens commun avec lesquels vous êtes habitués. Les expressions numériques entières sans autre préfixe sont considérées comme étant au format decimal.

Exemple:

```
101    // pareil que 101 decimal   ((1 * 100) + (0 * 10) + 1)
```

Format binaire

Le format binaire correspond à la base 2. Seuls les caractères 0 et 1 sont valides.

Exemple:

```
B101    // vaut 5 en décimal   ((1 * 2^2) + (0 * 2^1) + 1)
```

Le formateur binaire ne fonctionne uniquement sur des octets (soit 8 bits) entre 0 (B0) et 255 (B11111111). Si c'est nécessaire de saisir un int (16 bits) au format binaire, vous pouvez le faire en deux temps de la façon suivante :

```
myInt = (B11001100 * 256) + B10101010;    // B11001100 est l'octet de poids fort
```

Format octal

Le format octal correspond à la base 8. Seuls les caractères de 0 à 7 sont valides. Les valeurs octales sont indiquées par le préfix "0" Exemple:

```
0101    // vaut 65 en décimal   ((1 * 8^2) + (0 * 8^1) + 1)
```

ATTENTION : Il est possible de générer un bogue très difficile à détecter en incluant (par inadvertance) un 0 devant une expression numérique que le compilateur interprétera comme une expression numérique de type octal.

Format hexadécimal

Le format hexadécimal (or hex) correspond à la base 16. Les caractères valides sont de 0 à 9 ainsi que les lettres A à F; A a la valeur 10, B la valeur 11, jusqu'à F qui a la valeur 15. Les valeurs hexadécimales sont indiquées par le préfix "0x". Noter que les lettres A-F peuvent être écrites en majuscules(A-F) ou en minuscules (a-f).

Exemple:

```
0x101    // vaut 257 décimal   ((1 * 16^2) + (0 * 16^1) + 1)
```

3/ Types des données

Les variables peuvent être de type variés qui sont décrits ci-dessous.

- **boolean**
- **char**
- **byte**
- **int**
- **unsigned int**
- **long**
- **float** (nombres à virgules)
- **double** (nombres à virgules)
- **Les chaînes de caractères**
- **Les tableaux de variables**
- **void** (fonctions)

Description : boolean

Déclare une variable de type **boolean** (booléen ou binaire en anglais) qui ne peut prendre que deux valeurs : true ou false, VRAI ou FAUX, 1 ou 0, HIGH ou LOW (toutes ces façons de le dire sont équivalentes). (A noter que chaque variable de type **boolean** occupe cependant un octet de mémoire et non pas 1 bit.) **Syntaxe**

```
boolean  ma_variable = false; // déclare une variable de type binaire
```

Description : char

Déclare une variable de un octet de mémoire (8 bits) qui contient une valeur correspondant à un caractère. Les caractères unitaires sont écrits entre guillements uniques, comme ceci 'A' (pour des chaînes de caractères, utiliser les guillemets double : "ABC").

Syntaxe : char monChar='B'; // *déclare une variable char*

Exemple

char myChar = 'A'; // *déclare une variable char initialisée avec la valeur A* char myChar = 65; // *expression équivalente car la valeur ASCII de A est 65*

Description : byte

Déclare une variable de type octet (8 bits) qui stocke un nombre entier non-signé, soit une valeur de 0 à 255.

Syntaxe :

byte mon_byte =128;

Exemple :

byte b = B10010; // *Déclare une variable un octet non signet contenant la valeur binaire 10010 (=18)*

Description : int

Déclare une variable de type **int** (pour integer, entier en anglais). Les variables de type int sont votre type de base pour le stockage de nombres, et ces variables stockent une valeur sur 2 octets. Elles peuvent donc stocker des valeurs allant de - 32 768 à 32 767 (valeur minimale de -2 exposant 15 et une valeur maximale de (2 exposant 15) -1). **Syntaxe**

int var = val;

var : le nom de votre variable de type int
val : la valeur d'initialisation de la variable

Exemple :

int ledPin = 13; // *déclare une variable de type int appelée LedPin et valant 13*

Description : Les tableaux de variables

Un tableau est une collection de variables qui sont accessibles àl'aide d'un numéro d'index. En programmation en langage C, langage sur lequel le langage Arduino est basé, les tableaux peuvent être compliqués, mais utilisé de simples tableaux est relativement simple.

Créer (Déclarer) un tableau

Toutes les méthodes suivantes sont des façons correctes de créer (déclarer) un tableau de variables.

int myInts[6]; int myPins[] = {2, 4, 8, 3, 6}; int mySensVals[6] = {2, 4, -8, 3, 2}; char message[6] = "hello";

Les éléments d'un tableau sont "zero indexés", ce qui veut dire, si l'on se reporte aux initialisations de tableau ci-dessus, que le premier élément du tableau est à l'index 0. Ainsi :

mySensVals[0] == 2, mySensVals[1] == 4, et ainsi de suite

Cela signifie également que dans un tableau de 10 éléments, l'index 9 est les dernier élément. Ainsi :

int myArray[10]={9,3,2,4,3,2,7,8,9,11}; // *myArray[9] contient 11*

Description : void (fonctions)

Le mot-clé **void** est utilisé uniquement pour les déclarations de fonctions. Il indique au compilateur que l'on s'attend à ce que la fonction ne retourne aucune donnée à la fonction qui l'a appelée.

III- Fonctions :

1/ Entrées/Sorties Numériques

- **pinMode**(broche, mode) **Description :**

Configure la broche spécifiée pour qu'elle se comporte soit en entrée, soit en sortie. Voir la description des broches numériques pour plus de détails.

Syntaxe :

pinMode(broche, mode)

Paramètres : broche: le numéro de la broche de la carte Arduino dont le mode de fonctionnement (entrée ou sortie) doit être défini.

mode: soit INPUT (entrée en anglais) (=0) ou OUTPUT (sortie en anglais) (=1)

Valeur retournée :

Aucune

- **digitalWrite**(broche, valeur) **Description :**

Met un niveau logique HIGH (HAUT en anglais) ou LOW (BAS en anglais) sur une broche numérique. Si la broche a été configurée en SORTIE avec l'instruction pinMode(), sa tension est mise à la valeur correspondante : 5V (ou 3.3V sur les cartes Arduino 3.3V) pour le niveau HAUT, 0V (masse) pour le niveau BAS.

Syntaxe : digitalWrite(broche, valeur)

Paramètres : broche: le numéro de la broche de la carte Arduino **valeur** : HIGH ou LOW (ou bien 1 ou 0)

int **digitalRead**(broche) **Description:**

Lit l''état (= le niveau logique) d'une broche précise en entrée numérique, et renvoie la valeur HIGH (HAUT en anglais) ou LOW (BAS en anglais).

Syntaxe :

digitalRead(broche)

Paramètres :

Broche : le numéro de la broche numérique que vous voulez lire. (int) **Valeur retournée :**

Renvoie la valeur HIGH (HAUT en français) ou LOW (BAS en francais)

Exemple :

```
int ledPin = 13; // LED connectée à la broche n°13
int inPin = 7;   // un bouton poussoir connecté à la broche 7avec une résistance de pulldown int
val = 0;     // variable pour mémoriser la valeur lue

void setup()
{   pinMode(ledPin, OUTPUT);      // configure la broche 13 en SORTIE
pinMode(inPin, INPUT);      // configure la broche 7 en ENTREE
digitalWrite(inPin, HIGH);       // écrit la valeur HIGH (=1) sur la broche en entrée
                        // ce qui active la résistance de "rappel au +" (pullup) au plus de la broche
}
void loop()
{
  val = digitalRead(inPin);   // lit l'état de la broche en entrée et met le résultat dans la variable
   digitalWrite(ledPin, val);    // met la LED dans l'état du BP (càd allumée si appuyé et inversement)
}
```

2/ Entrées / Sortie analogiques

- **int** **analogRead**(broche)

Description:

Lit la valeur de la tension présente sur la broche spécifiée. La carte Arduino comporte 6 voies (8 voies sur la Mini et la Nano) connectées à un convertisseur analogique-numérique 10 bits. Cela signifie qu'il est possible de transformer la tension d'entrée entre 0 et 5V en une valeur numérique entière comprise entre 0 et 1023. Il en résulte une résolution (écart entre 2 mesures) de : 5 volts / 1024 intervalles, autrement dit une précision de 0.0049 volts (4.9 mV) par intervalle !
Une conversion analogique-numérique dure environ 100 µs (100 microsecondes soit 0.0001 seconde) pour convertir l'entrée analogique, et donc la fréquence maximale de conversion est environ de 10 000 fois par seconde.

Syntaxe :

```
analogRead(broche_analogique)
```

Paramètres :

broche_analogique : le numéro de la broche analogique (et non le numéro de la broche numérique) sur laquelle il faut convertir la tension analogique appliquée (0 à 5 sur la plupart des cartes Arduino, 0 à 7 sur la Mini et la Nano)

Exemple :

```
int analogPin = 3;     // une résistance variable (broche du milieu)
                       //connectée sur la broche analogique 3
                    // les autres broches de la résistance sont connectées
                    // l'une au 5V et l'autre au 0V
int val = 0; // variable de type int pour stocker la valeur de la mesure

void setup()
{
  Serial.begin(9600);          //  initialisation de la connexion série
// IMPORTANT : la fenêtre terminal côté PC doit être réglée sur la même valeur.
}
void loop()
{
  // lit la valeur de la tension analogique présente sur la broche
  val = analogRead(analogPin);
   // affiche la valeur (comprise en 0 et 1023) dans la fenêtre terminal
PC   Serial.println(val);              }
```

Description : analogWrite

Génère une impulsion de largeur / période voulue sur une broche de la carte Arduino (onde PWM - Pulse Width Modulation en anglais ou MLI - Modulation de Largeur d'Impulsion en français). Ceci peut-être utilisé pour faire briller une LED avec une luminosité variable ou contrôler un moteur à des vitesses variables.

Syntaxe

```
analogWrite(broche, valeur);
```

Paramètres :

broche: la broche utilisée pour "écrire" l'impulsion. Cette broche devra être une broche ayant la fonction PWM, Par exemple, sur la UNO, ce pourra être une des broches 3, 5 ,6 ,9 ,10 ou 11.

valeur: la largeur du "duty cycle" (proportion de l'onde carrée qui est au niveau HAUT) : entre 0 (0% HAUT donc toujours au niveau BAS) et 255 (100% HAUT donc toujours au niveau HAUT).

Exemple :

Fixer la luminosité d'une LED proportionnellement à la valeur de la tension lue depuis un potentiomètre.

```
int ledPin = 9;      // LED connectée sur la broche 9
int analogPin = 3;   // le potentiomètre connecté sur la broche analogique
3  int val = 0;      // variable pour stocker la valeur de la tension lue
void setup()
{
  pinMode(ledPin, OUTPUT);   // configure la broche en sortie
}
void loop()
{   val = analogRead(analogPin);   // lit la tension présente sur la broche en
entrée
  analogWrite(ledPin, val / 4);  // Résultat d'analogRead entre 0 to 1023,
                                 // résultat d'analogWrite entre 0 to
255                                  // => division par 4 pour
adaptation }
```

Temps :

- unsigned long millis()
- unsigned long micros()
- delay(ms)
- delayMicroseconds(us)

Math :

- min(x, y)
- max(x, y)
- abs(x)
- constrain(x, a, b)
- map(valeur, toLow, fromHigh, toLow, toHigh)
- pow(base, exposant)
- sq(x)
- sqrt(x) **Trigonométrie :** sin(rad)
- cos(rad)

- tan(rad)
- degrees(rad)
- radians(deg)
- PI

Description : **delay**(ms)

Réalise une pause dans l'exécution du programme pour la durée (en millisecondes) indiquée en paramètre. (=Pour mémoire, il y a 1000 millisecondes dans une seconde...!

Syntaxe : delay (ms)**;**

Paramètres :

ms (unsigned long): le nombre de millisecondes que dure la pause

Librairie Serial pour la communication série

1. Description

La librairie Serial est utilisée pour les communications par le port série entre la carte Arduino et un ordinateur ou d'autres composants. Toutes les cartes Arduino ont au moins un port Série (également désigné sous le nom de UART ou USART) : Serial. Ce port série communique sur les broches 0 (RX) et 1 (TX) avec l'ordinateur via le port USB. C'est pourquoi, si vous utilisez cette fonctionnalité, vous ne pouvez utiliser les broches 0 et 1 en tant qu'entrées ou sorties numériques.

Vous pouvez utiliser le terminal série intégré à l'environnement Arduino pour communiquer avec une carte Arduino. Il suffit pour cela de cliquer sur le bouton du moniteur série dans la barre d'outils puis de sélectionner le même débit de communication que celui utilisé dans l'appel de la fonction begin().

La carte Arduino Mega dispose de trois port série supplémentaires : Serial1 sur les broches 19 (RX) et 18 (TX), Serial2 sur les broches 17 (RX) et 16 (TX), Serial3 sur les broches 15 (RX) et 14 (TX).

2. Les fonctions de la librairie

- begin()
- available()
- read()
- flush()
- print()
- println()
- write()

1) Serial.begin(int vitesse)

DESCRIPTION

- Fixe le débit de communication en nombre de caractères par seconde (l'unité est le baud) pour la communication série.
- Pour communiquer avec l'ordinateur, utiliser l'un de ces débits : 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. Vous pouvez, également,

spécifier d'autres débits - par exemple, pour communiquer sur les broches 0 et 1 avec un composant qui nécessite un débit particulier.

SYNTAXE

Serial.begin(debit);

PARAMÈTRES

- int **debit**: debit de communication en caractères par seconde (ou baud). Pour communiquer avec l'ordinateur, utiliser l'un de ces débits : 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200.
- En pratique utiliser une valeur comprise entre 9600 et 115200. Typiquement 115200 fonctionne très bien ! (Plus le débit est élevé et plus la communication est rapide...)
- En cas d'utilisation du terminal série, IL FAUDRA FIXER LE MEME DEBIT dans la fenêtre du Terminal !

VALEUR RENVOYEE

Aucune

Exemple :

```
void setup() {
      Serial.begin(9600);     // ouvre le port série et fixe le débit de communication à 9600 bauds
}

void loop() {}
// Arduino Mega example

// Arduino Mega utilise l'ensemble de ses 4 ports série // (Serial, Serial1, Serial2, Serial3),
// avec des débits différents:

void setup(){
  Serial.begin(9600); // initialise le 1er port à 9600 bauds
  Serial1.begin(38400); // initialise le 2ème port série à 38400 bauds
  Serial2.begin(19200); // initialise le 3ème port série à 19200 bauds   Serial3.begin(4800); // initialise le 4ème port série à 4800 bauds

  Serial.println("Hello Computer"); // affiche un message via le 1er port série
  Serial1.println("Hello Serial 1"); // affiche un message via le 2ème port série
 Serial2.println("Hello Serial 2"); // affiche un message via le 3ème port série
Serial3.println("Hello Serial 3"); // affiche un message via le 4ème port série }
```

```
void loop() {}
```

2) int Serial.available()

DESCRIPTION

Donne le nombre d'octets (caractères) disponible pour lecture dans la file d'attente (buffer) du port série.

SYNTAXE

Serial.available();

PARAMÈTRES

Aucun

VALEUR RENVOYÉE

Le 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.

EXEMPLE

```
int incomingByte = 0;   // variable de stockage des données entrantes
 void setup() {
       Serial.begin(9600);     // ouvre le port série, fixe le débit à 9600 bauds }
 void loop() {
        // envoie des données seulement quand vous recevez des données :         if
(Serial.available() > 0) { // si des données entrantes sont présentes
              // lit le 1er octet arrivé
              incomingByte = Serial.read();
               // dit ce que vous obtenez
              Serial.print("J'ai recu : ");
              Serial.println(incomingByte, DEC);
       }
}
// Arduino Mega example:
 void setup() {   Serial.begin(9600);
  Serial1.begin(9600);
 }
 void loop()
{
```

```
 // lit du port 0, envoie du port 1
 if (Serial.available())
{
int inByte = Serial.read();
Serial1.print(inByte, BYTE);
  }
 // lit du port 1, envoie du port 0
 if (Serial1.available()) {     int inByte = Serial1.read();     Serial.print(inByte, BYTE);
 }
}
```

3) int Serial.read()

DESCRIPTION

Lit les données entrantes sur le port Série.

SYNTAXE

Serial.read();

PARAMÈTRES

Aucun

VALEUR RENVOYÉE

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

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

Exemple: Lit les données entrantes sur le port Série.

```
int incomingByte = 0;   // variable pour lecture de l'octet entrant
 void setup() {
       Serial.begin(9600);     // ouvre le port série et fixe le débit à 9600 bauds
}
 void loop() {
        // envoie une donnée sur le port série seulement quand reçoit une donnée          if (Serial.available() > 0) { // si données disponibles sur le port série
              // lit l'octet entrant                incomingByte = Serial.read();
               // renvoie l'octet reçu
```

```
            Serial.print("Octet recu: ");
            Serial.println(incomingByte, DEC);
        }
}
```

4) Serial.print(données) Description

Affiche les données sur le port série.

Affiche les données sous le port série sous forme lisible pour les humains (texte ASCII). Cette instruction peut prendre plusieurs formes.

- Les nombres entiers sont affichés en utilisant les caractères ASCII pour chaque chiffre.
- Les nombres à virgules (float) sont affichés de la même façon sous forme de caractères ASCII pour chaque chiffre, par défaut avec 2 décimales derrière la virgule.
- Les valeurs de type byte sont affichés sous la forme d'un caractère ASCII.
- Les caractères et les chaînes sont affichés tels que.

Par exemple :

```
Serial.print(78); // affiche "78"
Serial.print(1.23456); // affiche "1.23"
Serial.print(byte(78)); // affiche "N" (dont la valeur ASCII est 78)
Serial.print('N'); // affiche "N"
Serial.print("Hello world."); // affiche "Hello world."
```

Un second paramètre optionnel (format) spécifie :

- pour les nombres entiers, la base à utiliser. Les valeurs autorisées sont BYTE, BIN (binaire, ou base 2), OCT (octal, ou base 8), DEC (décimal, ou base 10), HEX (hexadécimal, ou base 16),
- pour les nombres à virgules (float), le paramètre précise le nombre de décimales après la virgule à utiliser.

Par exemple :

```
Serial.print(78, BYTE); // affiche "N"
Serial.print(78, BIN) ; // affiche  "1001110"
Serial.print(78, OCT); // affiche "116"
Serial.print(78, DEC); // affiche "78"
Serial.print(78, HEX); // affiche "4E"
Serial.print(1.23456, 0); // affiche "1"
Serial.print(1.23456, 2); // affiche "1.23"
Serial.print(1.23456, 4); // affiche "1.2346"
```

Syntaxe de base Serial.print(val)

```
Serial.print(val, format)
```

Paramètre de base val: la valeur à afficher. N'importe

quel type de données.

format : spécifie la base utilisée(pour les nombres entiers) ou le nombre de décimales (pour les nombres de type float)

Valeur renvoyée

Aucune

Syntaxe détaillée

Cette commande peut prendre de nombreuses formes :

Serial.print(b) avec aucun format spécifié, affiche b en tant que nombre decimal sous la forme d'une chaîne de caractère ASCII. Par exemple,

```
int b = 79;
```

Serial.print(b)**;** affiche la chaîne de caractère "79".

Serial.print(b, DEC) affiche b en tant que nombre décimal sous la forme d'une chaine de caractères ASCII.

Par exemple :

```
int b = 79;
Serial.print(b, DEC); affiche la chaîne de caractères "79".
Serial.print(b, HEX) affiche b en tant que nombre hexadécimal sous la forme d'une chaîne de caractère ASCII. Par exemple :
int b = 79;
Serial.print(b, HEX); affiche la chaîne de caractères "4F".
Serial.print(b, OCT) affiche b en tant que nombre octal sous la forme d'une chaîne de caractère ASCII. Par exemple :
int b = 79;
Serial.print(b, OCT); affiche la chaîne de caractères "117".
Serial.print(b, BIN) affiche b en tant que nombre binairesous la forme d'une chaîne de caractère ASCII.
```

Par exemple :

```
int b = 79;
Serial.print(b, BIN); affiche la chaîne de caractères "1001111".
```

Serial.print(b, BYTE) affiche b en tant que simple octet, ce qui est interprété comme un caractère ASCII. Par exemple :

int b = 79;

Serial.print(b, BYTE); renvoie le caractère "O", ce caractère ASCII étant représenté par la valeur 79. Pour plus d'informations, voir le Code ASCII.

Serial.print(str) si str est une chaîne ou un tableau de variables char, affiche str sous la forme d'une chaîne de caractère ASCII. Par exemple :

Serial.print("Hello World!"); affiche la chaîne de caractère "Hello World!".

Paramètres détaillés

- **b**: l'octet à afficher, ou
- **str**: la chaîne à afficher

Exemple

```
/*Utilise une boucle FOR pour afficher un nombre dans des formats variés*/ int x = 0;    // variable

void setup() {
  Serial.begin(9600);      // ouvre le port série à 9600 bauds
}
void loop() {
  // affiche une entête de présentation en colonnes
  Serial.print("SANS FORMAT");       // affiche le texte
  Serial.print("\t");              // affiche un tab
  Serial.print("DEC");
  Serial.print("\t");
  Serial.print("HEX");
  Serial.print("\t");
  Serial.print("OCT");
  Serial.print("\t");
  Serial.print("BIN");
  Serial.print("\t");
  Serial.println("BYTE"); // affiche le texte suivi d'un saut de ligne
  for(x=0; x< 64; x++){    // affiche une partie de la table
ASCII
    // affichage sous différents formats
    Serial.print(x);       // affiche un decimal en chaîne
```

```
ASCII- comme "DEC"
   Serial.print("\t");    // affiche un tab
   Serial.print(x, DEC);  // affiche un décimal en chaîne ASCII
   Serial.print("\t");    // affiche un tab
   Serial.print(x, HEX);  // affiche un hexadecimal en chaîne
ASCII
   Serial.print("\t");    // affiche un tab
   Serial.print(x, OCT);  // affiche un octal en chaîne ASCII
   Serial.print("\t");    // affiche un tab
   Serial.print(x, BIN);  // affiche un binaire en chaîne ASCII
   Serial.print("\t");    // affiche un tab
   Serial.println(x, BYTE);    // affiche un octet,  //et ajoute le retour du chariot avec
"println"
   delay(200);           // pause de 200 millisecondes
  }
 Serial.println("");      // affiche un autre retour de chariot
```

Serial.println(data)

DESCRIPTION

Affiche les données sur le port série suivi d'un caractère de "retour de chariot" (ASCII 13, or '\r') et un caractère de "nouvelle ligne" (ASCII 10, or '\n'). Cette instruction a par ailleurs la même forme que l'instruction Serial.print():

Serial.println(b) affiche b en tant que nombre décimal sous la forme d'une chaîne de caractères ASCII suivi d'un retour de chariot et d'un saut de ligne.
Serial.println(b, DEC) affiche b en tant que nombre décimal sous la forme d'une chaîne de caractères ASCII suivi d'un retour de chariot et d'un saut de ligne.
Serial.println(b, HEX) affiche b en tant que nombre hexadécimal sous la forme d'une chaîne de caractères ASCII suivi d'un retour de chariot et d'un saut de ligne.
Serial.println(b, OCT) affiche b en tant que nombre octal sous la forme d'une chaîne de caractères ASCII suivi d'un retour de chariot et d'un saut de ligne.
Serial.println(b, BIN) affiche b en tant que nombre binaire sous la forme d'une chaîne de caractères ASCII suivi d'un retour de chariot et d'un saut de ligne.
Serial.print(b, BYTE) affiche b en tant qu'octet simple suivi d'un retour de chariot et d'un saut de ligne.
Serial.println(str) si str est une chaîne de caractère ou un tableau de caractère, affiche la chaîne de caractère suivie d'un retour de chariot et d'un saut de ligne. **Serial.println()** affiche un retour de chariot et un saut de ligne.

PARAMÈTRES

- **data** : tous types de données entières incluant les char, chaînes de caractères et floats (nombre à virgule).

Les floats sont supportés avec une précision de 2 à plusieurs décimales.

VALEUR RENVOYÉE

Aucune

EXEMPLE

```
/*Lit une valeur analogique en entrée sur analog 0 et affiche la valeur. */
int analogValue = 0;    // variable pour stocker la valeur analogique
void setup() {
 // initialise le port série à 9600 bauds
 Serial.begin(9600);
}
void loop() {
 // lit la valeur analogique sur la broche analog 0   analogValue = analogRead(0);
 // affiche cette valeur dans différents formats
 Serial.println(analogValue);       // affichage décimal
 Serial.println(analogValue, DEC);  // affichage décimal
 Serial.println(analogValue, HEX);  // affichage hexadécimal
 Serial.println(analogValue, OCT);  // affichage octal
 Serial.println(analogValue, BIN);  // affichage binaire
 Serial.println(analogValue, BYTE); // affichage octet simple
 // pause de 10 millisecondes avant la mesure suivante   delay(10);
}
```