**eBook Gratuit**

# APPRENEZ arduino

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#arduino

# Table des matières

# À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: arduino

It is an unofficial and free arduino ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official arduino.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Chapitre 1: Démarrer avec Arduino

## Remarques

# Qu'est ce que Arduino?

Arduino est une plate-forme électronique open source basée sur du matériel et des logiciels faciles à utiliser.

# Pourquoi utiliser Arduino?

- Peu coûteux. Vous pouvez également acheter des clones encore moins chers.
- Facile à utiliser et commencez avec
- Communauté énorme
- Complètement Open Source

## Versions

| Version | Date de sortie |
| --- | --- |
| 1.0.0 | 2016-05-08 |

## Examples

### Le strict minimum

Voici l'esquisse Arduino "au strict minimum". Cela peut être chargé dans l'IDE Arduino en choisissant `File > Examples > 01. Basics > Bare Minimum .`

```
void setup() {
  // put your setup code here, to run once
}

void loop() {
  // put your main code here, to run repeatedly
}
```

Le code de la fonction `setup()` sera exécuté une fois au démarrage du programme. Ceci est utile pour configurer des broches d'E / S, initialiser des variables, etc. Le code dans la fonction `loop()` sera exécuté à plusieurs reprises jusqu'à ce que l'Arduino soit désactivé ou qu'un nouveau programme soit téléchargé. Effectivement, le code ci-dessus ressemble à ceci dans la bibliothèque d'exécution Arduino:

```
setup();
while(1) {
  loop();
}
```

Contrairement aux programmes exécutés sur votre ordinateur, le code Arduino ne peut jamais être arrêté. Ceci est dû au fait que le microcontrôleur ne contient qu'un seul programme. Si ce programme se ferme, il n'y aura rien à dire au microcontrôleur.

## Cligner

Voici un court exemple qui illustre les fonctions `setup()` et `loop()` . Cela peut être chargé dans l'IDE Arduino en choisissant `File > Examples > 01. Basics > Blink` . ( *Remarque: la* plupart des cartes Arduino ont une LED déjà connectée à la broche 13, mais vous devrez peut-être ajouter une LED externe pour voir les effets de cette esquisse.)

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

L'extrait ci-dessus:

1. Définit la fonction `setup()` . La fonction `setup()` est appelée en premier sur l'exécution dans chaque programme Arduino.

    1. Définit la broche 13 en sortie.

        Sans cela, il pourrait être réglé sur une entrée, ce qui rendrait la LED non fonctionnelle. Cependant, une fois défini en tant que sortie, il restera comme cela, il ne faut donc le faire qu'une fois au démarrage du programme.

2. Définit la fonction `loop()` . La fonction `loop()` est appelée à plusieurs reprises tant que le programme est en cours d'exécution.

    1. `digitalWrite(13, HIGH);` allume la LED.
    2. `delay(1000);` attend une seconde (1 000 millisecondes).
    3. `digitalWrite(13, LOW);` éteint la LED.
    4. `delay(1000);` attend une seconde (1000 millisecondes).

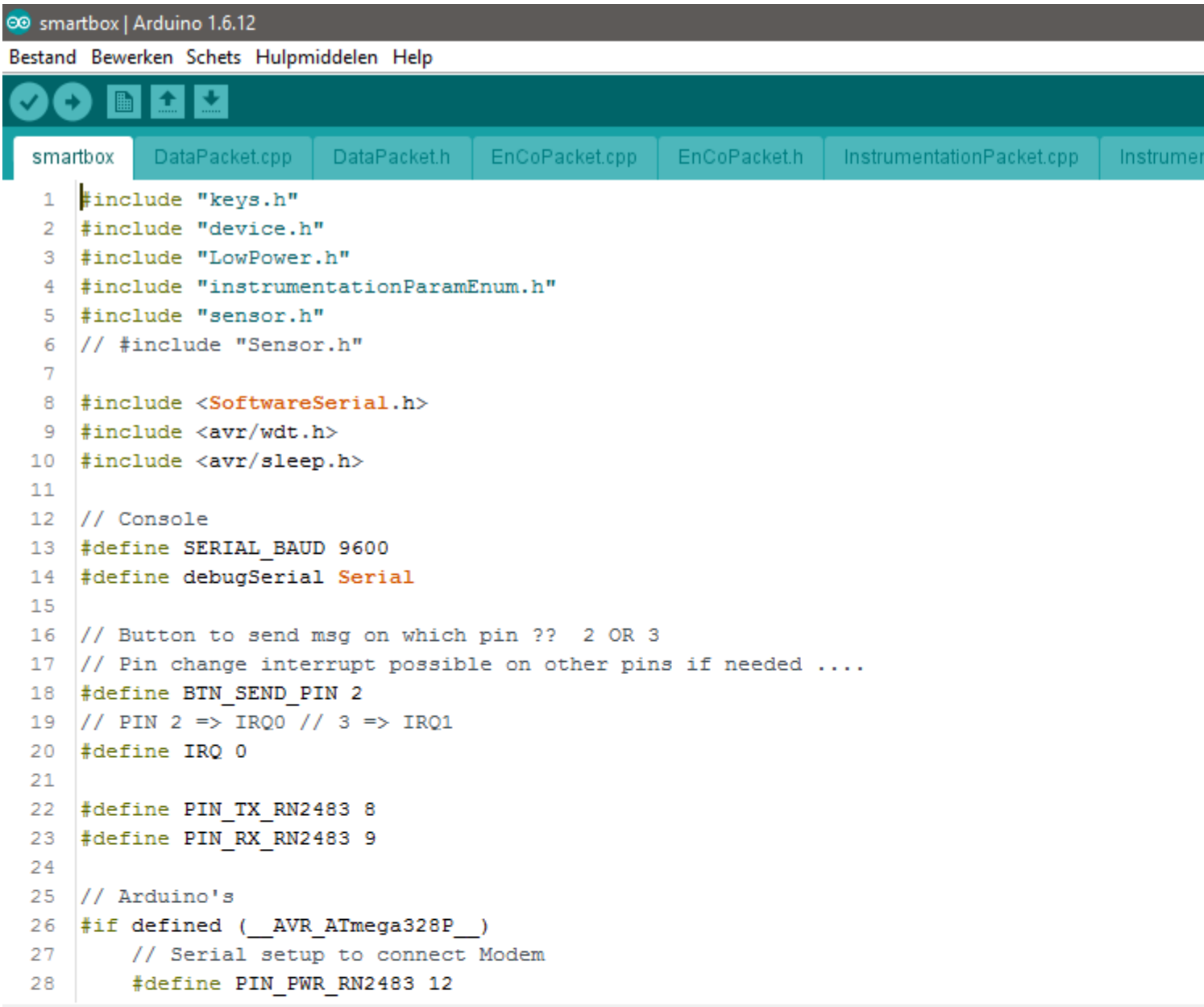
Étant donné que `loop()` est exécuté de manière répétée tant que le programme est en cours d'exécution, le voyant clignote pendant 2 secondes (1 seconde, 1 seconde). Cet exemple est basé

sur l'Arduino Uno et sur toute autre carte ayant déjà une DEL connectée à la broche 13. Si la carte utilisée n'a pas de DEL intégrée connectée à cette broche, l'une d'entre elles peut être connectée en externe.

Plus sur le timing (par exemple, les délais et le temps de mesure): gestion du temps

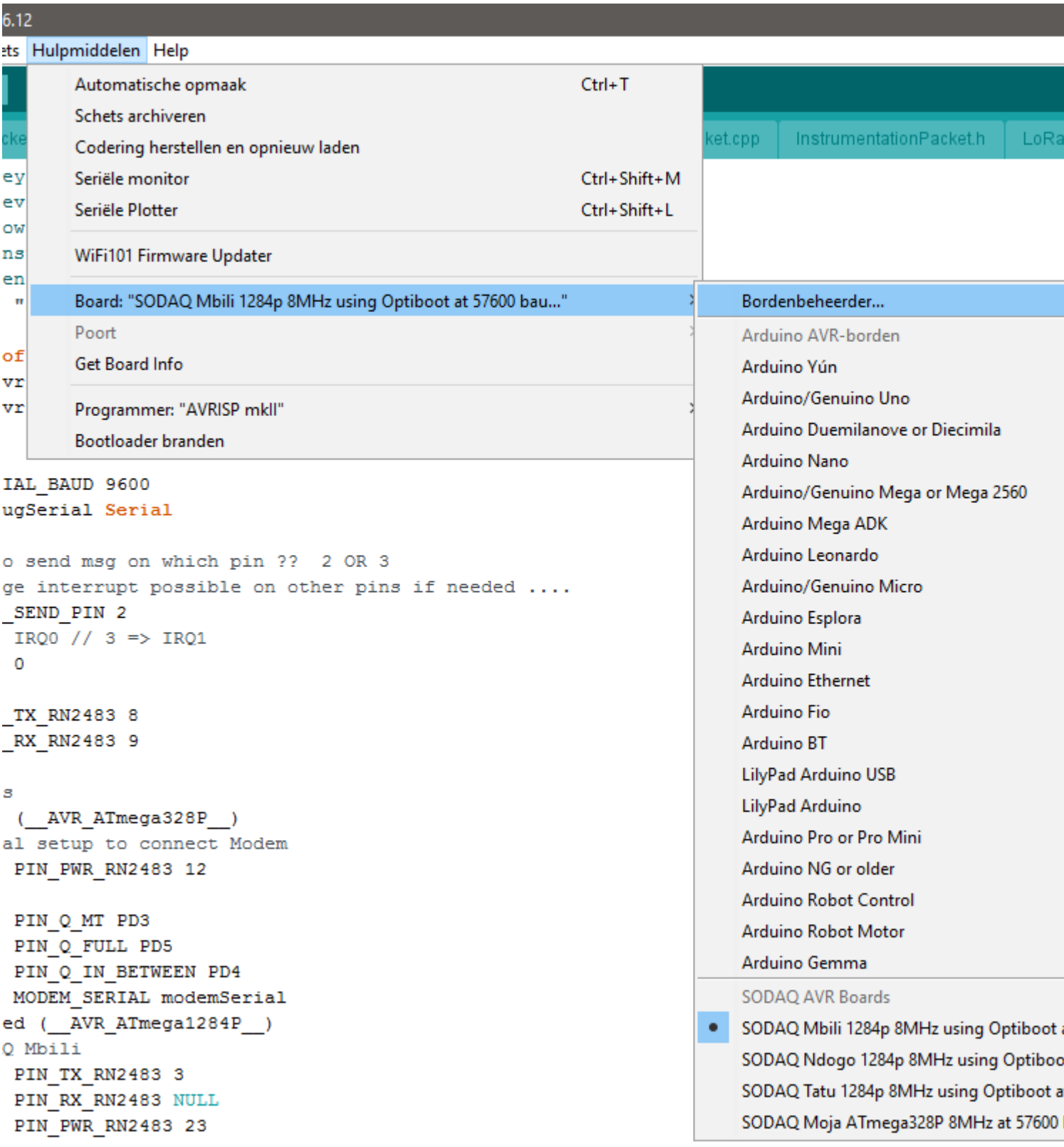
## Première installation

Logiciel requis: IDE Arduino


smartbox | Arduino 1.6.12
Bestand Bewerken Schets Hulpmiddelen Help
smartbox
DataPacket.cpp
DataPacket.h
EnCoPacket.cpp
EnCoPacket.h
InstrumentationPacket.cpp
#include "keys.h"
#include "device.h"
#include "LowPower.h"
#include "instrumentationParamEnum.h"
#include "sensor.h"
// #include "Sensor.h"
#include <SoftwareSerial.h>
#include <avr/wdt.h>
#include <avr/sleep.h>
// Console
#define SERIAL_BAUD 9600
#define debugSerial Serial
// Button to send msg on which pin ??  2 OR 3
// Pin change interrupt possible on other pins if needed ....
#define BTN_SEND_PIN 2
// PIN 2 => IRQ0 // 3 => IRQ1
#define IRQ 0
#define PIN_TX_RN2483 8
#define PIN_RX_RN2483 9
// Arduino's
#if defined (__AVR_ATmega328P__)
// Serial setup to connect Modem
#define PIN_PWR_RN2483 12
"DataPacket.h" contains unrecognized characters. If this code was created with an older versi
"LoRaModem.h" contains unrecognized characters. If this code was created with an older versio

`setup()` et `loop()` . Ceci est suffisant pour télécharger sur une carte Arduino, mais il ne fera rien du tout. L'exemple d'esquisse "Blink" fonctionne comme un test simple lors de la première utilisation d'une carte Arduino. Allez dans Fichier → Exemples → 01.Basics → Blink. Cela ouvrira une nouvelle fenêtre avec l'esquisse Blink.

Sélectionnez votre tableau Allez dans Outils → Panneau → [nom de votre carte Arduino].




Sélectionnez le port COM de votre carte. La plupart des cartes compatibles Aurduino créeront un

faux port COM, utilisé pour la communication série (débogage) et pour la programmation de la carte. COM 1 est *généralement* déjà présent et votre forum en créera un nouveau, par exemple COM 4. Sélectionnez ceci dans Outils → Port → COM 4 (ou un autre numéro COM).

Certaines cartes ont des paramètres supplémentaires dans le menu Outils, tels que la vitesse d'horloge. Celles-ci varient d'un conseil à l'autre, mais généralement, un ensemble de valeurs par défaut acceptable est déjà sélectionné.

# Télécharger

Vous êtes maintenant prêt à télécharger Blink. Cliquez sur le bouton Télécharger ou sélectionnez Esquisse → Télécharger. Le croquis sera compilé, puis téléchargé sur votre carte Arduino. Si tout fonctionnait bien, la DEL intégrée clignotera toutes les secondes.


# Moniteur série

Dans l'IDE Arduino, vous avez un moniteur série. Pour l'ouvrir, utilisez le *moniteur série* à droite de la fenêtre.


Assurez-vous que le code est téléchargé avant d'ouvrir le moniteur. Le téléchargement et le moniteur ne fonctionneront pas en même temps!

## LED - Avec contrôle de bouton

Vous pouvez également utiliser ce code pour configurer une LED avec un interrupteur à bouton avec une résistance de tirage, cela pourrait être de préférence avec l'étape suivante après la configuration du contrôleur de LED initial.

```
int buttonState = 0; // variable for reading the pushbutton status

void setup()
{
    // initialize the LED pin as an output:
    pinMode(13, OUTPUT); // You can set it just using its number
    // initialize the pushbutton pin as an input:
    pinMode(2, INPUT);
}

void loop()
{
    // read the state of the pushbutton value:
    buttonState = DigitalRead(2);
```

```
    // check if the pushbutton is pressed.
    // If it's not, the buttonState is HIGH : if (buttonState == HIGH)
    {
        // turn LED off:
        digitalWrite(13, LOW);
    }
    else
    {
        // turn LED off:
        digitalWrite(13, HIGH);
    }
}
```

Lire Démarrer avec Arduino en ligne: https://riptutorial.com/fr/arduino/topic/610/demarrer-avec-arduino

# Chapitre 2: Arduino IDE

## Examples

### Installation sous Windows

1. Allez à https://www.arduino.cc/en/Main/Software
2. Cliquez sur le lien "Windows Installer"
3. Suivez les instructions

### Application portable sous Windows

Pour utiliser l'IDE Arduino sous Windows sans avoir besoin de l'installer:

1. Allez à https://www.arduino.cc/en/Main/Software
2. Cliquez sur le lien "Fichier Windows ZIP pour installation non admin"
3. Extraire l'archive dans un dossier
4. Ouvrez le dossier et double-cliquez sur `Arduino.exe`

### Installation sur Fedora

1. Ouvrez un terminal et lancez: `sudo dnf install arduino`
2. Ouvrez l'application Arduino ou tapez `arduino` dans le terminal

### Installation sur Ubuntu

1. Ouvrez un terminal et lancez: `sudo apt-get install arduino`
2. Ouvrez l'application Arduino ou tapez `arduino` dans le terminal

### Installation sur macOS

1. Allez à https://www.arduino.cc/en/Main/Software
2. Cliquez sur le lien `Mac OS X`
3. Décompressez le fichier `.zip` .
4. Déplacez l'application `Arduino` vers `Applications` .

Lire Arduino IDE en ligne: https://riptutorial.com/fr/arduino/topic/3790/arduino-ide

# Chapitre 3: Bibliothèque de cristaux liquides

## Introduction

La `Liquid Crystal Library` Arduino est une bibliothèque pour contrôler les écrans LCD compatibles avec le pilote Hitachi HD44780, caractérisé par son interface 16 broches. Les 16 broches peuvent être connectées via une interface I2C. Ces écrans contiennent une matrice de blocs de 5x7 pixels utilisés pour afficher des caractères ou de petites images monochromatiques. Les affichages sont généralement nommés en fonction du nombre de lignes et de colonnes dont ils disposent, par exemple 16x2 ou 1602 pour 16 colonnes et 2 lignes et 20x4 ou 2004 pour 20 colonnes et 4 lignes.

## Syntaxe

- #include <LiquidCrystal.h> // Inclut la bibliothèque
- LiquidCrystal (rs, enable, d4, d5, d6, d7) //
- LiquidCrystal (rs, rw, enable, d4, d5, d6, d7)
- LiquidCrystal (rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)
- LiquidCrystal (rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)

## Paramètres

| Paramètre LiquidCrystal | Détails |
|---|---|
| rs | le numéro de la broche Arduino connectée à la broche RS sur l'écran LCD |
| rw | le numéro de la broche Arduino connectée à la broche RW de l'écran LCD (facultatif) |
| activer | le numéro de la broche Arduino connectée à la broche d'activation sur l'écran LCD |
| d0 - d7 | les numéros des broches Arduino connectées aux broches de données correspondantes sur l'écran LCD. d0, d1, d2 et d3 sont facultatifs; En cas d'omission, l'écran LCD sera contrôlé en utilisant uniquement les quatre lignes de données (d4, d5, d6, d7). |

## Examples

### Utilisation de base

```
/*
```

```
  Wiring:
   LCD pin 1 (VSS) -> Arduino Ground
   LCD pin 2 (VDD) -> Arduino 5V
   LCD pin 3 (VO)  -> Arduino Ground
   LCD pin 4 (RS)  -> Arduino digital pin 12
   LCD pin 5 (RW)  -> Arduino Ground
   LCD pin 6 (E)   -> Arduino digital pin 11
   LCD pin 11 (D4) -> Arduino digital pin 5
   LCD pin 12 (D5) -> Arduino digital pin 4
   LCD pin 13 (D6) -> Arduino digital pin 3
   LCD pin 14 (D7) -> Arduino digital pin 2
*/

#include <LiquidCrystal.h> // include the library

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // start writing on the first row and first column.
  lcd.setCursor(0, 0);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // No need to do anything to keep the text on the display
}
```

Lire Bibliothèque de cristaux liquides en ligne:
https://riptutorial.com/fr/arduino/topic/9395/bibliotheque-de-cristaux-liquides

# Chapitre 4: Bibliothèques

## Introduction

**Vous trouverez ici la documentation sur:**

-Installer des bibliothèques dans l'IDE Arduino

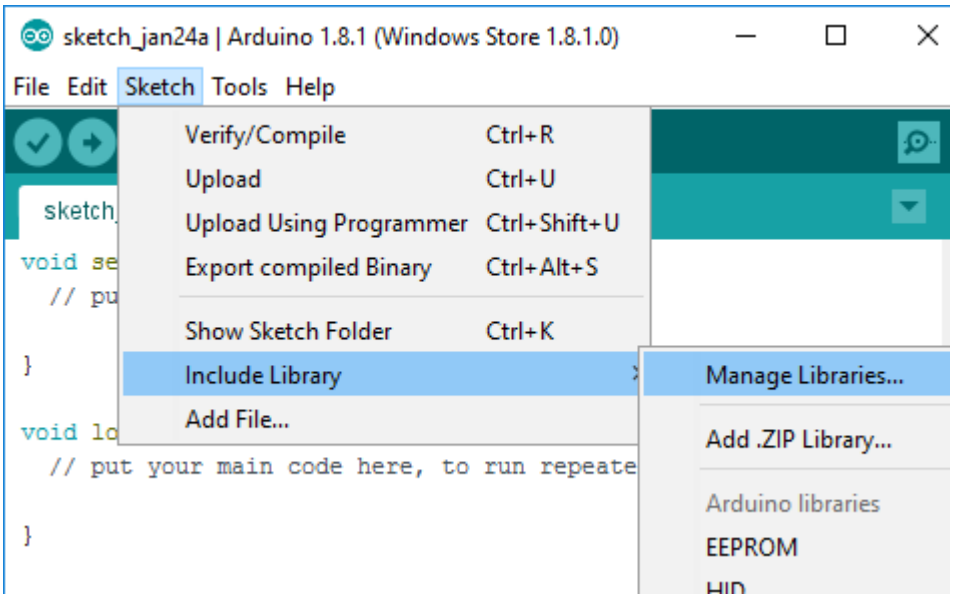
-Inclusion des bibliothèques dans un croquis

## Examples

### Installation des bibliothèques avec le gestionnaire de bibliothèque

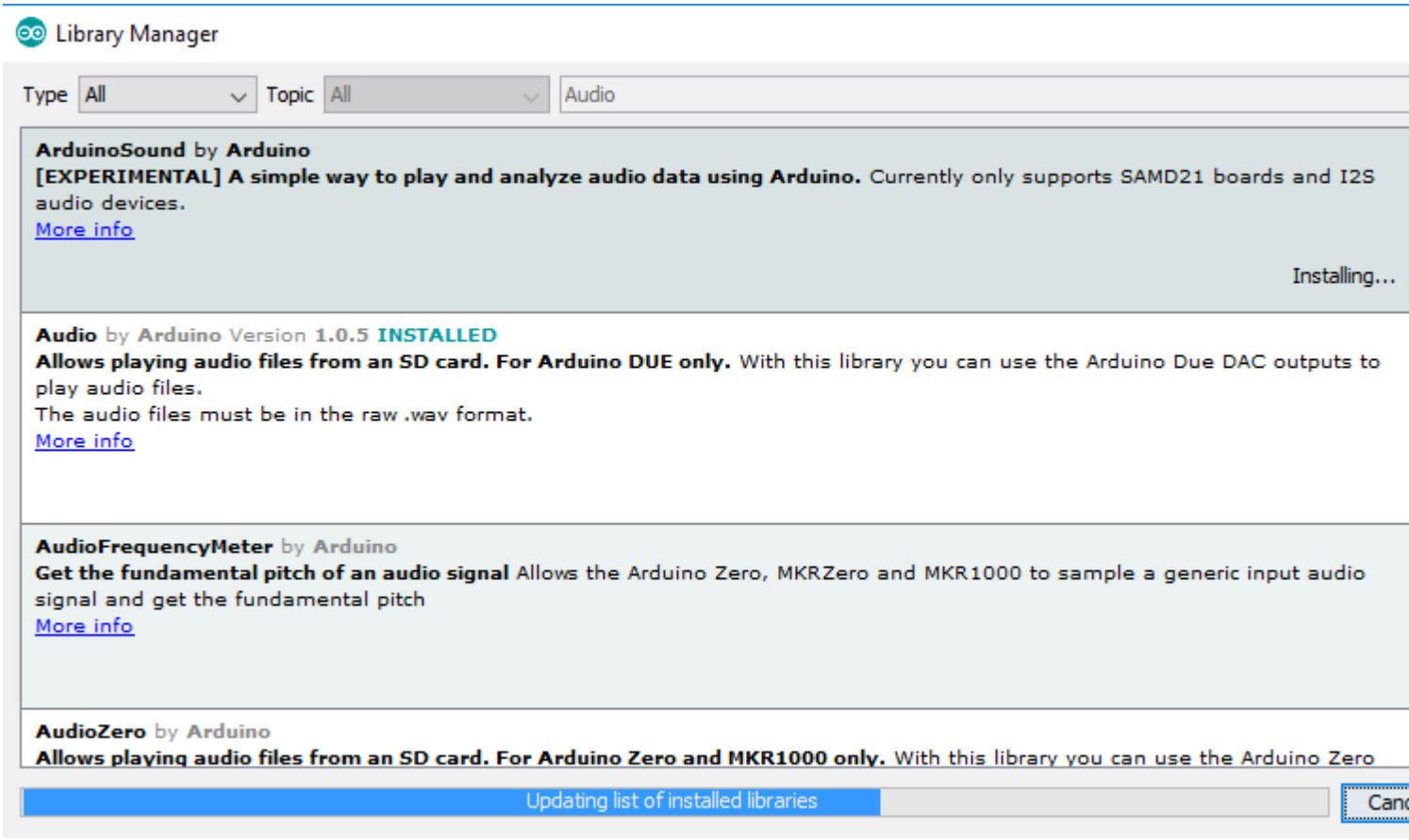
Pour installer une nouvelle bibliothèque dans l'IDE Arduino:

- **Ouvrir le menu Esquisse> Inclure la bibliothèque> Gérer les bibliothèques.**




*Une fois que vous avez ouvert le gestionnaire de bibliothèque, vous pouvez utiliser le menu en haut pour filtrer les résultats.*

- **Cliquez sur la bibliothèque de votre choix, sélectionnez une version dans le menu déroulant, puis cliquez sur installer.**

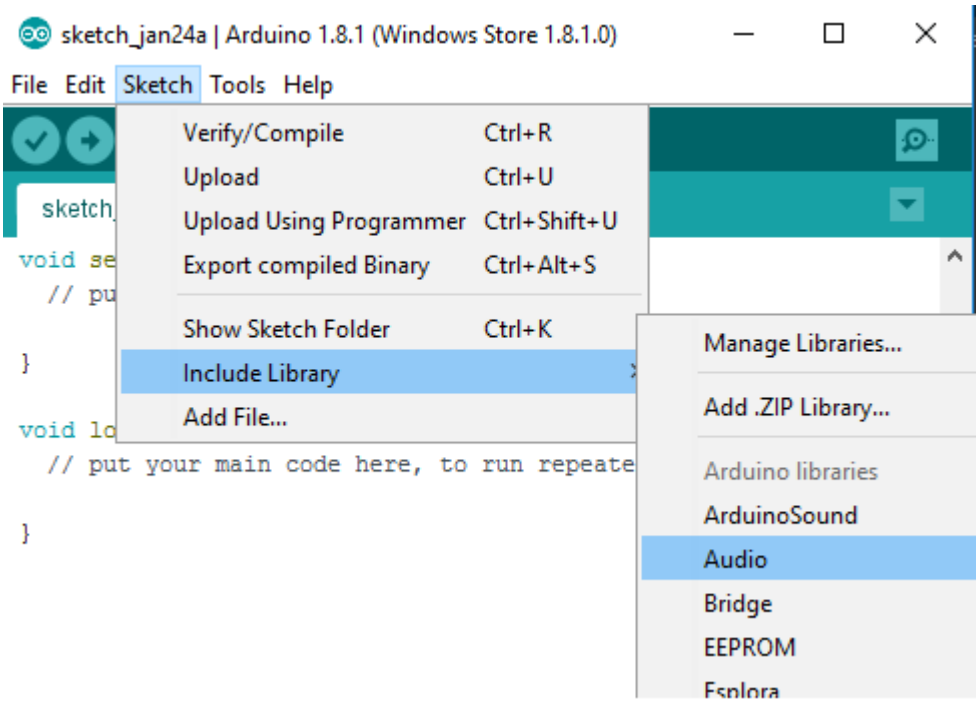



Votre bibliothèque est maintenant installée. Pour l'utiliser, vous devez l'inclure dans votre croquis.

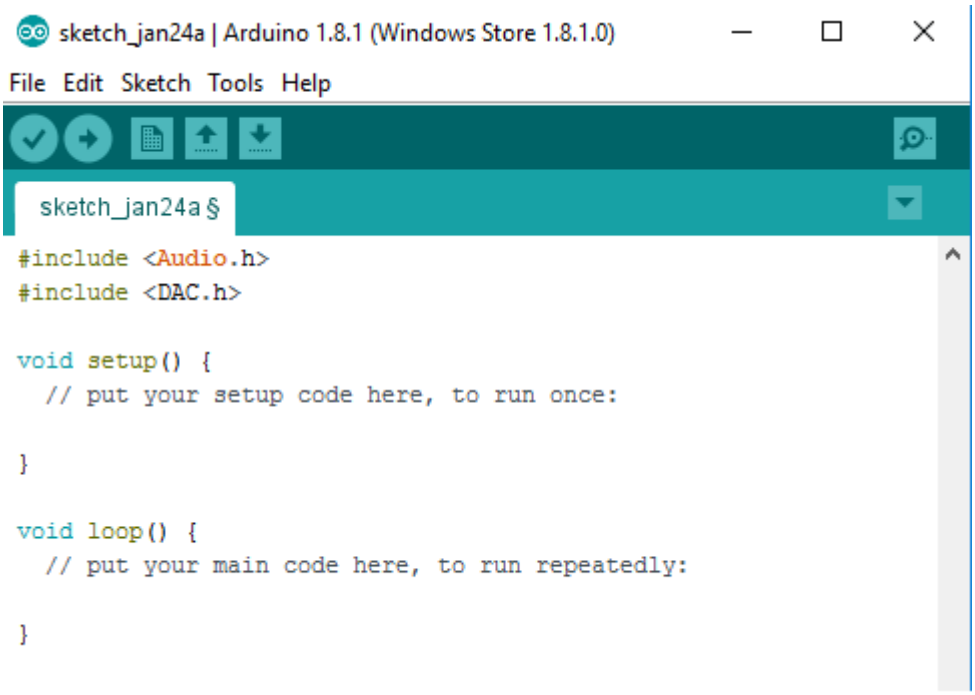
## Y compris les bibliothèques dans votre esquisse.

Une fois que vous avez installé une bibliothèque, vous devez l'inclure dans votre esquisse pour pouvoir l'utiliser.

- **Ouvrez le menu Esquisse> Inclure la bibliothèque et cliquez sur la bibliothèque que vous souhaitez inclure.**




- **Maintenant, l'EDI a généré les balises d'inclusion requises dans votre code.**




La bibliothèque est maintenant incluse dans votre esquisse et vous pouvez l'utiliser dans votre code.

Lire Bibliothèques en ligne: https://riptutorial.com/fr/arduino/topic/8896/bibliotheques

# Chapitre 5: Boucles

## Syntaxe

- pour (déclaration, condition, itération) {}
- while (condition) {}
- do {} while (condition)

## Remarques

`General Remark` Si vous avez l'intention de créer une boucle pour attendre que quelque chose se produise, vous êtes probablement sur la mauvaise voie ici. Rappelez-vous plutôt que tout le code après setup () est exécuté à partir d'une méthode appelée loop (). Donc, si vous avez besoin d'attendre quelque chose, c'est plus facile de ne rien faire (ou seulement d'autres choses indépendantes) et de revenir pour vérifier la condition d'attente la prochaine fois.

`do { } while(condition)` n'évaluera pas l'instruction de condition avant la première itération. Il est important de garder cela à l'esprit si la déclaration de condition a des effets secondaires.

## Examples

### Tandis que

Une `while` boucle évaluera son état, et si `true` , il exécutera le code à l' intérieur et recommencer. Autrement dit, tant que sa condition est évaluée à `true` , le `while` en boucle exécutera plus et plus.

---

Cette boucle sera exécutée 100 fois, en ajoutant chaque fois 1 à la variable `num` :

```
int num = 0;
while (num < 100) {
    // do something
    num++;
}
```

La boucle ci-dessus est équivalente à une boucle `for` :

```
for (int i = 0; i < 100; i++) {
    // do something
}
```

---

Cette boucle s'exécutera pour toujours:

```
while (true) {
    // do something
}
```

La boucle ci-dessus est équivalente à une boucle `for` :

```
for (;;) {
    // do something
}
```

## Pour

`for` les boucles sont syntaxe simplifiée pour une configuration de boucle très fréquent, ce qui pourrait être accompli en plusieurs lignes avec un `while` boucle.

Voici un exemple courant de boucle `for` , qui s'exécutera 100 fois puis s'arrêtera.

```
for (int i = 0; i < 100; i++) {
    // do something
}
```

Cela équivaut à une `while` boucle:

```
int num = 0;
while (num < 100) {
    // do something
    num++;
}
```

Vous pouvez créer une boucle sans fin en omettant la condition.

```
for (;;) {
    // do something
}
```

Cela équivaut à une `while` boucle:

```
while (true) {
    // do something
}
```

## Faire pendant

A `do while` boucle est le même que `while` la boucle, sauf qu'il est garanti d'exécuter au moins une fois.

La boucle suivante sera exécutée 100 fois.

```
int i = 0;
do {
    i++;
} while (i < 100);
```

Une boucle similaire, mais avec une condition différente, exécutera 1 fois.

```
int i = 0;
do {
    i++;
} while (i < 0);
```

Si la boucle au- dessus était simplement un `while` boucle, il exécute 0 fois, parce que la condition évaluerait à `false` avant la première itération. Mais comme il s'agit d'une boucle `do while` while, elle s'exécute une fois, puis vérifie sa condition avant de l'exécuter à nouveau.

## Contrôle de flux

Il existe des moyens de briser ou de modifier le flux d'une boucle.

`break;` quittera la boucle en cours et n'exécutera plus de lignes dans cette boucle.

`continue;` n'exécutera plus de code dans l'itération courante de la boucle, mais restera dans la boucle.

La boucle suivante exécutera 101 fois (i = 0, 1, ..., 100) au lieu de 1000 en raison de l'instruction `break` :

```
for (int i = 0; i < 1000; i++) {
    // execute this repeatedly with i = 0, 1, 2, ...
    if (i >= 100) {
        break;
    }
}
```

La boucle suivante aura pour résultat que la valeur de `j` sera 50 au lieu de 100, à cause de l'instruction `continue` :

```
int j=0;
for (int i = 0; i < 100; i++) {
    if (i % 2 == 0) { // if `i` is even
        continue;
    }
    j++;
}
// j has the value 50 now.
```

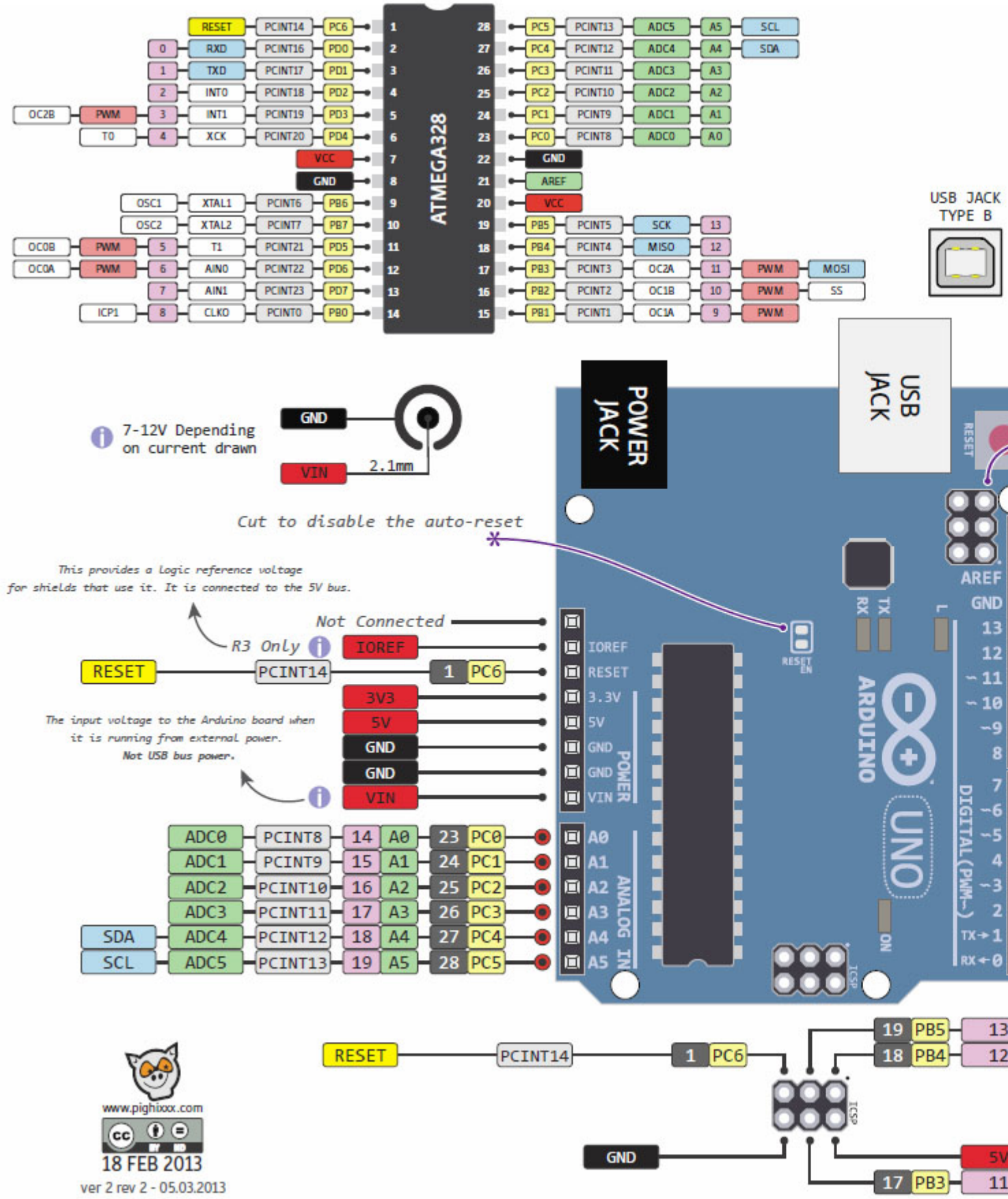
Lire Boucles en ligne: https://riptutorial.com/fr/arduino/topic/2802/boucles

# Chapitre 6: Broches de matériel

## Examples

### Arduino Uno R3

Les microcontrôleurs utilisent des broches pour interagir avec le reste du circuit. Ces broches seront généralement l'une des broches d'entrée / sortie, vin ou terre. Les broches d'E / S peuvent être de simples broches d'E / S numériques ou présenter certaines caractéristiques spécifiques, telles que la possibilité de faire varier la tension de leur sortie en utilisant une modulation de largeur d'impulsion. Voici un schéma de l'Arduino R3 Uno et de ses broches.




( source )

**Pins PWM**

PWM vous permet de contrôler la tension de la sortie en commutant la sortie entre haute et basse très très rapidement. Le pourcentage de temps pendant lequel la broche est haute est appelé «cycle de travail».

Broches PWM: 3, 5, 6, 9, 10, 11

**Entrées analogiques**

Tout comme une broche PWM peut émettre une gamme de tensions, les broches analogiques de l'Arduino Uno R3 peuvent détecter une gamme de tensions d'entrée. Vous pouvez l'utiliser pour lire la position d'un potentiomètre ou d'une autre entrée avec une entrée variable en douceur. Veuillez noter que les broches analogiques ne peuvent pas effectuer de sortie analogWrite - pour cela, vous devez utiliser des broches PWM.

Broches analogiques du CAN: A0, A1, A2, A3, A4, A5

**Série, SPI et I2C**

Les broches série de l'Arduino Uno R3 sont également utilisées par exemple par la puce USB vers série lorsqu'elle communique avec un ordinateur via le port USB intégré. Série: Tx sur 0, Rx sur 1

SPI et I2C sont des protocoles de communication que l’Arduino peut utiliser pour communiquer avec des blindages, des capteurs, des sorties, etc.:

Pins SPI: MOSI on 11, MISO 12, SCLK 13, SS 10

Broches I2C: SCL sur A5, SDA sur A4

**LED embarquée**

L’Arduino Uno R3 possède une LED avec sa propre résistance connectée à la broche 13. Cela signifie que même si vous n’attachez aucune LED à votre carte, si vous réglez la broche 13 sur une sortie et que vous la définissez à un niveau élevé, vous devriez voir une LED sur le tableau venez. Utilisez l'exemple d'esquisse «Blink» pour localiser votre LED intégrée.

De la page des épingles numériques d'Arduino

> REMARQUE: La broche numérique 13 est plus difficile à utiliser comme entrée numérique que les autres broches numériques car elle est connectée à la carte sur la plupart des cartes par une LED et une résistance. Si vous activez sa résistance de tirage interne de 20k, celle-ci sera maintenue à environ 1,7V au lieu de 5V, car les résistances LED et série intégrées réduisent le niveau de tension, ce qui signifie qu’elle restitue toujours LOW. Si vous devez utiliser la broche 13 comme entrée numérique, réglez son mode pinMode () sur INPUT et utilisez une résistance de rappel externe.

Broche LED embarquée: 13

Lire Broches de matériel en ligne: https://riptutorial.com/fr/arduino/topic/4386/broches-de-materiel

# Chapitre 7: Comment Python s'intègre à Arduino Uno

## Syntaxe

- `Serial.begin(baudrate) // Set baud rate (bits per second) for serial data transmission`
- `Serial.println(value) // Print data to serial port followed by Carriage Return \r and Newline character \n`
- `serial.Serial((port=None, baudrate=9600, bytesize=EIGHTBITS, parity=PARITY_NONE, stopbits=STOPBITS_ONE, timeout=None, xonxoff=False, rtscts=False, write_timeout=None, dsrdtr=False, inter_byte_timeout=None) // Initialize serial port with all parameters`
- `serial.readline() // Read serial data which contains Carriage Return \r and Newline character \n`

## Paramètres

| Paramètre | Détails |
|---|---|
| en série | Le paquet Python contient des classes et des méthodes pour accéder au port série |
| temps | Le package Python inclut des fonctions liées au temps |

## Remarques

J'utilise un Arduino Uno avec Arduino IDE 1.6.9 et Python 2.7.12 fonctionnant sous Windows 10.

## Examples

### Première communication série entre Arduino et Python

Dans ce premier exemple, une opération d'écriture série de base est lancée à partir d'un périphérique Arduino.

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.println("Hello World!");
  delay(100);
}
```

Dans `setup()` , la fonction `Serial.begin(9600)` configure le débit en bauds pour la communication de

données en série. Dans cet exemple, un débit en bauds de 9600 est utilisé. D'autres valeurs peuvent être lues ici: Fonction Arduino Serial.begin ()

Dans `loop()` , le premier message que nous souhaitons envoyer est "Hello World!". Ce message est transmis en utilisant `Serial.println("Hello World!")` il enverra cette chaîne au port série au format ASCII. À la fin du message, il y a le retour de chariot `(CR, \r)` et le caractère de nouvelle ligne `(\n)` . De plus, un délai de 100 millisecondes est utilisé chaque fois que le programme imprime sur le port série.

Ensuite, téléchargez cette esquisse Arduino via le port COM (rappelez-vous ce numéro de port COM tel qu'il sera utilisé dans le programme Python).

Le programme Python qui lit les données série envoyées par l'appareil Arduino est illustré ci-dessous:

```
import serial
import time

ser = serial.Serial('COM8', 9600)
while (1):
    print ser.readline()
    time.sleep(0.1)
```

Tout d'abord, le paquetage pyserial doit être importé. Pour plus d'informations sur l'installation de pyserial dans un environnement Windows, veuillez consulter cette instruction: Installation de Python et de pyserial . Ensuite, nous initialisons le port série avec le numéro de port COM et le débit en bauds. Le débit en bauds doit être identique à celui utilisé dans l'esquisse Arduino.

Le message reçu sera imprimé en boucle en utilisant la fonction `readline()` . Un délai de 100 millisecondes est également utilisé ici, comme dans l'esquisse Arduino. Veuillez noter que la fonction `readline()` pyserial nécessite un délai d'expiration lors de l'ouverture d'un port série (documentation pyserial : PySerial ReadLine ).

Lire Comment Python s'intègre à Arduino Uno en ligne:
https://riptutorial.com/fr/arduino/topic/6722/comment-python-s-integre-a-arduino-uno

# Chapitre 8: Comment stocker des variables dans EEPROM et les utiliser pour le stockage permanent

## Syntaxe

- EEPROM.write (adresse, valeur); // (Stocker les variables dans EEPROM dans une adresse particulière)
- EEPROM.read (adresse); // (Récupère les valeurs de l'EEPROM et lit les données stockées dans l'EEPROM)

## Paramètres

| Paramètres de EEPROM.write | Détail |
|---|---|
| adresse | L'adresse où la valeur doit être stockée dans l'EEPROM |
| valeur | Variable principale à stocker dans l'EEPROM. Notez qu'il s'agit d'un `uint_8` (single byte) - vous devez séparer vous-même les types de données à plusieurs octets en octets simples. Vous pouvez également utiliser `EEPROM.put` pour stocker des flottants ou d'autres types de données. |
| **Paramètres d'EEPROM.Lire** | Détail |
| adresse | L'adresse à partir de laquelle la variable doit être lue |

## Remarques

Les adresses autorisées varient selon le matériel.

- ATMega328 (Uno, Pro Mini, etc.): 0–1023
- ATMega168: 0-511
- ATMega1280: 0-4095
- ATMega2560: 0-4095

la source

## Examples

## Stockez une variable dans EEPROM, puis récupérez-la et imprimez-la sur l'écran

Tout d'abord, ajoutez une référence à `<EEPROM.h>` au début de votre esquisse:

```
#include <EEPROM.h>
```

Ensuite, votre autre code:

```
// Stores value in a particular address in EEPROM. There are almost 512 addresses present.

    // Store value 24 to Address 0 in EEPROM
    int addr = 0;
    int val = 24;
    EEPROM.write(addr, val);     // Writes 24 to address 0

    // ---------
    // Retrieves value from a particular address in EEPROM
    // Retrieve value from address 0 in EEPROM
    int retrievedVal = EEPROM.read(0);    // Retrieves value stored in 0 address in
                                          // EEPROM

    // *[NOTE: put Serial.begin(9600); at void setup()]*
    Serial.println(retrievedVal);        // Prints value stored in EEPROM Address 0 to
                                         // Serial (screen)
```

Lire Comment stocker des variables dans EEPROM et les utiliser pour le stockage permanent en ligne: https://riptutorial.com/fr/arduino/topic/5987/comment-stocker-des-variables-dans-eeprom-et-les-utiliser-pour-le-stockage-permanent

# Chapitre 9: Communication Bluetooth

## Paramètres

| méthode | détails |
|---|---|
| SoftwareSerial.h | Documentation |
| SoftwareSerial (rxPin, txPin, inverse_logic) | Constructeur. **rxPin** : La valeur par **défaut des** données dans la broche (de réception) est 0. **txPin** : **Broche de** sortie de données (transmise), valeur par défaut: 1. **inverse_logic** : si true, traite LOW comme si HIGH et HIGH. par défaut à false. |
| commencer (vitesse) | Définit le débit en bauds pour la communication série. Les débits en bauds pris en charge sont 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400, 57600 et 115200. |
| disponible() | Vérifiez s'il y a des données sur la série |
| lis() | Lit une chaîne de série |
| écoute() | Vérifie si le port série du logiciel demandé écoute activement. |
| débordement() | Vérifie si un dépassement de tampon série du logiciel s'est produit. L'appel de cette fonction efface l'indicateur de débordement, ce qui signifie que les appels suivants renverront false sauf si un autre octet de données a été reçu et éliminé entre-temps. Le tampon série du logiciel peut contenir 64 octets. |
| peek () | Renvoie un caractère reçu sur la broche RX du port série du logiciel. Contrairement à read (), les appels suivants à cette fonction renverront le même caractère. Notez qu'une seule instance de SoftwareSerial peut recevoir des données entrantes à la fois (sélectionnez celle avec la fonction `listen()` ). |
| imprimer (données) | Imprime les données sur la broche de transmission du port série du logiciel. Fonctionne de la même manière que la fonction `Serial.print()` . |
| println (données) | Imprime les données sur la broche de transmission du port série du logiciel, suivie d'un retour chariot et d'un saut de ligne. Fonctionne de la même manière que la fonction `Serial.println()` . |
| écouter () | Permet au port série du logiciel sélectionné d'écouter. Un seul port série de logiciel peut écouter à la fois; les données qui arrivent pour les autres ports seront supprimées. Toutes les données déjà reçues sont ignorées pendant l'appel à `listen()` (à moins que l'instance |

| méthode | détails |
| --- | --- |
| | donnée n'écoute déjà). |
| écrire (données) | Imprime les données sur la broche de transmission du port série du logiciel en tant qu'octets bruts. Fonctionne comme la fonction `Serial.write()` . |

# Remarques

Erreur commune: Si vous conservez les broches rx et tx à leurs valeurs par défaut (0 et 1), vous ne pouvez pas télécharger de nouveau code tant que vous ne le supprimez pas.

# Examples

## Monde de base bluetooth bonjour

```
#include <SoftwareSerial.h>
// its always better to change the default tx and rx as the may interfere with other process
in future.

// configure tx , rx by defualt they will be 0 and 1 in arduino UNO
SoftwareSerial blue(3,2);
void setup() {
  // preferred baud rate/data transfer rate in general is 38400
     blue.begin(38400);
  // do initialization or put one time executing code here
}

void loop() {

  // put code that you want it to run every time no matter what
    if(blue.available()){
        // put only that code which needsd to run when there is some data
        // This means that the their is some data sent over the bluetooth
        // You can do something with the data

        int n;
        // consider that the data received to be integer, read it by using blue.parseInt();

        n = blue.parseInt();

    }

}
```

Lire Communication Bluetooth en ligne:
https://riptutorial.com/fr/arduino/topic/2543/communication-bluetooth

# Chapitre 10: Communication I2C

## Introduction

I2C est un protocole de communication qui permet à deux cartes Arduino ou plus de se parler. Le protocole utilise deux broches - SDA (ligne de données) et SCL (ligne d'horloge). Ces broches sont différentes d'un type de carte Arduino à un autre, vérifiez donc les spécifications de la carte. Le protocole I2C définit une carte Arduino comme maître et toutes les autres comme esclave. Chaque esclave a une adresse différente que le programmeur a définie en dur. Remarque: Assurez-vous que toutes les cartes connectées à la même source VCC

## Examples

### Plusieurs esclaves

L'exemple suivant montre comment le maître peut recevoir des données de plusieurs esclaves. Dans cet exemple, l'esclave envoie deux numéros courts. Le premier est pour la température et le second pour l'humidité. Veuillez noter que la température est un flotteur (24.3). Pour utiliser seulement deux octets et non quatre (float est quatre octets), je multiplie la température par 10 et la sauvegarde en tant que court-circuit. Voici donc le code maître:

```
#include <Wire.h>

#define BUFFER_SIZE  4
#define MAX_NUMBER_OF_SLAVES 24
#define FIRST_SLAVE_ADDRESS 1
#define READ_CYCLE_DELAY 1000

byte buffer[BUFFER_SIZE];

void setup()
{
  Serial.begin(9600);
  Serial.println("MASTER READER");
  Serial.println("**************");

  Wire.begin();        // Activate I2C link
}

void loop()
{
  for (int slaveAddress = FIRST_SLAVE_ADDRESS;
       slaveAddress <= MAX_NUMBER_OF_SLAVES;
       slaveAddress++)
  {
    Wire.requestFrom(slaveAddress, BUFFER_SIZE);    // request data from the slave
    if(Wire.available() == BUFFER_SIZE)
      {  // if the available data size is same as I'm expecting
        // Reads the buffer the slave sent
        for (int i = 0; i < BUFFER_SIZE; i++)
        {
```

```
          buffer[i] = Wire.read();  // gets the data
        }

        // Parse the buffer
        // In order to convert the incoming bytes info short, I use union
        union short_tag {
          byte b[2];
          short val;
        } short_cast;

        // Parse the temperature
        short_cast.b[0] = buffer[0];
        short_cast.b[1] = buffer[1];
        float temperature = ((float)(short_cast.val)) / 10;

        // Parse the moisture
        short_cast.b[0] = buffer[2];
        short_cast.b[1] = buffer[3];
        short moisture = short_cast.val;

        // Prints the income data
        Serial.print("Slave address ");
        Serial.print(slaveAddress);
        Serial.print(": Temprature = ");
        Serial.print(temprature);
        Serial.print("; Moisture = ");
        Serial.println(moisture);
      }
    }
    Serial.println("*************************");

    delay(READ_CYCLE_DELAY);
  }
}
```

Et maintenant le code de l'esclave:

```
#include <Wire.h>
#include <OneWire.h>
#include <DallasTemperature.h>

//=====================
// This is the hard-coded address. Change it from one device to another
#define SLAVE_ADDRESS 1
//=====================

// I2C Variables
#define BUFFER_SIZE 2
#define READ_CYCLE_DELAY 1000
short data[BUFFER_SIZE];

// Temprature Variables
OneWire oneWire(8);
DallasTemperature temperatureSensors(&oneWire);
float m_temperature;

// Moisture Variables
short m_moisture;

// General Variables
```

```
int m_timestamp;

void setup()
{
  Serial.begin(9600);
  Serial.println("SLAVE SENDER");
  Serial.print("Node address: ");
  Serial.println(SLAVE_ADDRESS);
  Serial.print("Buffer size: ");
  Serial.println(BUFFER_SIZE * sizeof(short));
  Serial.println("**********************");

  m_timestamp = millis();
  Wire.begin(NODE_ADDRESS);  // Activate I2C network
  Wire.onRequest(requestEvent); // Set the request event handler
  temperatureSensors.begin();
}

void loop()
{
  if(millis() - m_timestamp < READ_CYCLE_DELAY) return;

  // Reads the temperature
  temperatureSensors.requestTemperatures();
  m_temperature = temperatureSensors.getTempCByIndex(0);

  // Reads the moisture
  m_moisture = analogRead(A0);
}

void requestEvent()
{
  data[0] = m_temperature * 10; // In order to use short, I multiple by 10
  data[1] = m_moisture;
  Wire.write((byte*)data, BUFFER_SIZE * sizeof(short));
}
```

Lire Communication I2C en ligne: https://riptutorial.com/fr/arduino/topic/9092/communication-i2c

# Chapitre 11: Communication MIDI

## Introduction

L'intention de cette rubrique est de démontrer certains programmes MIDI de base qui montrent comment utiliser le protocole et ajouter progressivement des fonctionnalités utiles que les applications plus complexes requièrent.

## Examples

### MIDI THRU Exemple

Le MIDI Thru est simple et facile à tester. Lorsque vous travaillez correctement, vous pourrez installer votre projet Arduino entre deux appareils MIDI, MIDI IN sur MIDI OUT et vous pourrez vérifier que les deux appareils fonctionnent ensemble. Si vous avez la possibilité de mesurer la latence, vous verrez une augmentation due à la capture de mémoire tampon en série et aux instructions de retransmission.

```
// This is a simple MIDI THRU.  Everything in, goes right out.
// This has been validate on an Arduino UNO and a Olimex MIDI Shield

boolean byteReady;
unsigned char midiByte;

void setup() {
    // put your setup code here, to run once:
    //  Set MIDI baud rate:
    Serial.begin(31250);
    byteReady = false;
    midiByte = 0;
}

// The Loop that always gets called...
void loop() {
   if (byteReady) {
        byteReady = false;
        Serial.write(midiByte);
    }
}

// The little function that gets called each time loop is called.
// This is automated somwhere in the Arduino code.
void serialEvent() {
  if (Serial.available()) {
    // get the new byte:
    midiByte = (unsigned char)Serial.read();
    byteReady = true;
  }
}
```

### MIDI Thru with Queue

```
// This is a more complex MIDI THRU.  This version uses a queue.  Queues are important because
some
// MIDI messages can be interrupted for real time events.  If you are generating your own
messages,
// you may need to stop your message to let a "real time" message through and then resume your
message.


#define QUEUE_DEPTH 128

// Queue Logic for storing messages
int headQ = 0;
int tailQ = 0;
unsigned char tx_queue[QUEUE_DEPTH];

void setup() {
    // put your setup code here, to run once:
    //  Set MIDI baud rate:
    Serial.begin(31250);
}

// getQDepth checks for roll over.  Folks have told me this
// is not required.  Feel free to experiment.
int getQDepth() {
int depth = 0;
    if (headQ < tailQ) {
        depth = QUEUE_DEPTH - (tailQ - headQ);
    } else {
        depth = headQ - tailQ;
    }
    return depth;
}

void addQueue (unsigned char myByte) {
    int depth = 0;
    depth = getQDepth();

    if (depth < (QUEUE_DEPTH-2)) {
        tx_queue[headQ] = myByte;
        headQ++;
        headQ = headQ % QUEUE_DEPTH; // Always keep the headQ limited between 0 and 127
    }
}

unsigned char deQueue() {
    unsigned char myByte;
    myByte = tx_queue[tailQ];
    tailQ++;
    tailQ = tailQ % QUEUE_DEPTH;  // Keep this tailQ contained within a limit
    // Now that we dequeed the byte, it must be sent.
    return myByte;
}

void loop() {
   if (getQDepth>0) {
        Serial.write(deQueue());
    }
}

// The little function that gets called each time loop is called.
// This is automated somwhere in the Arduino code.
```

```
void serialEvent() {
  if (Serial.available()) {
    // get the new byte:
    addQueue((unsigned char)Serial.read());;
  }
}
```

## Génération d'horloge MIDI

```
// This is a MiDI clk generator.  This takes a #defined BPM and
// makes the appropriate clk rate.  The queue is used to let other messages
// through, but allows a clock to go immediately to reduce clock jitter

#define QUEUE_DEPTH 128
#define BPM 121
#define MIDI_SYSRT_CLK 0xF8

// clock tracking and calculation
unsigned long lastClock;
unsigned long captClock;
unsigned long clk_period_us;

// Queue Logic for storing messages
int headQ = 0;
int tailQ = 0;
unsigned char tx_queue[QUEUE_DEPTH];

void setup() {
    //  Set MIDI baud rate:
    Serial.begin(31250);
    clk_period_us = 60000000 / (24 * BPM);
    lastClock = micros();
}

// getQDepth checks for roll over.  Folks have told me this
// is not required.  Feel free to experiment.
int getQDepth() {
int depth = 0;
    if (headQ < tailQ) {
        depth = QUEUE_DEPTH - (tailQ - headQ);
    } else {
        depth = headQ - tailQ;
    }
    return depth;
}

void addQueue (unsigned char myByte) {
    int depth = 0;
    depth = getQDepth();

    if (depth < (QUEUE_DEPTH-2)) {
        tx_queue[headQ] = myByte;
        headQ++;
        headQ = headQ % QUEUE_DEPTH; // Always keep the headQ limited between 0 and 127
    }
}

unsigned char deQueue() {
    unsigned char myByte;
```

```
    myByte = tx_queue[tailQ];
    tailQ++;
    tailQ = tailQ % QUEUE_DEPTH;  // Keep this tailQ contained within a limit
    // Now that we dequeed the byte, it must be sent.
    return myByte;
}

void loop() {
    captClock = micros();

    if (lastClock > captClock) {
        // we have a roll over condition - Again, maybe we don't need to do this.
        if (clk_period_us <= (4294967295 - (lastClock - captClock))) {
            // Add a the ideal clock period for this BPM to the last measurement value
            lastClock = lastClock + clk_period_us;
            // Send a clock, bypasing the transmit queue
            Serial.write(MIDI_SYSRT_CLK);
        }
    } else if (clk_period_us <= captClock-lastClock) {
        // Basically the same two commands above, but not within a roll over check
        lastClock = lastClock + clk_period_us;
        // Send a clock, bypasing the transmit queue
        Serial.write(MIDI_SYSRT_CLK);
    }

    if (getQDepth>0) {
        Serial.write(deQueue());
    }
}

// The little function that gets called each time loop is called.
// This is automated somwhere in the Arduino code.
void serialEvent() {
  if (Serial.available()) {
    // get the new byte:
    addQueue((unsigned char)Serial.read());;
  }
}
```

## Messages MIDI définis

En général, le protocole MIDI est décomposé en "messages". Il existe 4 classes générales de messages:

- Voix Channel
- Mode de canal
- Système commun
- Messages système en temps réel

Les messages commencent par une valeur d'octet supérieure à 0x80. Toute valeur inférieure à 0x7F est considérée comme une donnée. Ce qui signifie que 127 est la valeur maximale pouvant être encodée dans un seul octet de données MIDI. Pour encoder des valeurs plus grandes, deux octets de données MIDI ou plus sont requis.

Il convient de signaler que les messages doivent être envoyés pour terminer sans interruption ... SAUF ... Système Les messages en temps réel, qui sont un seul octet, peuvent être injectés au

milieu de n'importe quel message.

**Messages vocaux de canal**

| Statut D7..D0 | Octets de données | La description |
|---|---|---|
| 1000nnnn | 0kkkkkkk 0vvvvvvv | Note Off event. Ce message est envoyé lorsqu'une note est libérée (terminée). (kkkkkkk) est le numéro de la clé (note). (vvvvvvv) est la vitesse. |
| 1001nnnn | 0kkkkkkk 0vvvvvvv | Remarque sur l'événement. Ce message est envoyé lorsqu'une note est enfoncée (début). (kkkkkkk) est le numéro de la clé (note). (vvvvvvv) est la vitesse. |
| 1010nnnn | 0kkkkkkk 0vvvvvvv | Pression de touche polyphonique (Aftertouch). Ce message est le plus souvent envoyé en appuyant sur la touche (kkkkkkk) est le numéro de la clé (note). (vvvvvvv) est la valeur de la pression. |
| 1011nnnn | 0ccccccc 0vvvvvvv | Changement de contrôle. Ce message est envoyé lorsqu'une valeur de contrôleur change. Les contrôleurs incluent des dispositifs tels que des pédales et des leviers. Les numéros de contrôleur 120-127 sont réservés en tant que "Messages de mode de canal" (ci-dessous). (ccccccc) est le numéro du contrôleur (0-119). (vvvvvvv) est la valeur du contrôleur (0-127). |
| 1100nnnn | 0ppppppp | Changement de programme. Ce message est envoyé lorsque le numéro de patch change. (ppppppp) est le nouveau numéro de programme. |
| 1101nnnn | 0vvvvvvv | Pression du canal (After-touch). Ce message est le plus souvent envoyé en appuyant sur la touche Ce message est différent du post-touch polyphonique. Utilisez ce message pour envoyer la valeur de pression maximale la plus élevée (de toutes les touches enfoncées actuelles). (vvvvvvv) est la valeur de la pression. |
| 1110nnnn | 0lllllll 0mmmmmmm | Changement de Pitch Bend. Ce message est envoyé pour indiquer une modification de la hauteur tonale (roue ou levier, en général). Le pitch bender est mesuré par une valeur de quatorze bits. Centre (pas de changement de hauteur) est 2000H. La sensibilité est une fonction du récepteur, mais peut être définie à l'aide de RPN 0. (lllllll) sont les 7 bits les moins significatifs. (mmmmmmm) sont les 7 bits les plus significatifs. |

**Messages du mode canal**

| Statut D7..D0 | Octets de données | La description |
|---|---|---|
| 1011nnnn | 0ccccccc 0vvvvvvv | Messages du mode canal. C'est le même code que le changement de commande (ci-dessus), mais implémente le contrôle de mode et le message spécial en utilisant les numéros de contrôleur réservés 120-127. Les commandes sont les suivantes: |
| | | Tout son éteint. À la réception de tous les sons désactivés, tous les oscillateurs s'éteignent et leurs enveloppes de volume sont mises à zéro dès que possible. c = 120, v = 0: tout son éteint |
| | | Réinitialiser tous les contrôleurs. Lorsque Réinitialiser tous les contrôleurs est reçu, toutes les valeurs du contrôleur sont réinitialisées à leurs valeurs par défaut. (Voir les pratiques spécifiques recommandées pour les défauts). |
| | | c = 121, v = x: la valeur ne doit être nulle que si cela est autorisé autrement dans une pratique spécifique recommandée. |
| | | Contrôle local. Lorsque le contrôle local est désactivé, tous les périphériques d'un canal donné répondent uniquement aux données reçues via MIDI. Les données lues, etc. seront ignorées. Local Control On restaure les fonctions des contrôleurs normaux. |
| | | c = 122, v = 0: contrôle local désactivé |
| | | c = 122, v = 127: Contrôle local activé |
| | | Toutes les notes sont désactivées. Lorsqu'un signal All Notes est reçu, tous les oscillateurs sont désactivés. |
| | | c = 123, v = 0: Toutes les notes sont désactivées (Voir le texte pour la description des commandes du mode réel.) |
| | | c = 124, v = 0: Mode Omni désactivé |
| | | c = 125, v = 0: mode omni activé |
| | | c = 126, v = M: Mode mono activé (Poly désactivé) où M est le nombre de canaux (Omni Off) ou 0 (Omni On) |
| | | c = 127, v = 0: Mode Poly activé (Mono Off) (Remarque: ces quatre messages provoquent également la désactivation de toutes les notes) |

**Messages communs au système**

| Statut D7..D0 | Octets de données | La description |
|---|---|---|
| 11110000 | 0iiiiiii [0iiiiiii 0iiiiiii] 0ddddddd --- --- 0ddddddd 11110111 | Système exclusif Ce type de message permet aux fabricants de créer leurs propres messages (tels que des vidages en masse, des paramètres de patch et d'autres données non spécifiées) et fournit un mécanisme pour créer des messages de spécification MIDI supplémentaires. Le code d'identification du fabricant (attribué par MMA ou AMEI) est soit 1 octet (0iiiiiii), soit 3 octets (0iiiiiii 0iiiiiii 0iiiiiii). Deux des ID à 1 octet sont réservés aux extensions appelées messages exclusifs universels, qui ne sont pas spécifiques à un fabricant. Si un appareil reconnaît le code d'identification comme étant le sien (ou comme un message universel pris en charge), il écoutera le reste du message (0ddddddd). Sinon, le message sera ignoré. (Remarque: seuls les messages en temps réel peuvent être entrelacés avec un système exclusif.) |
| 11110001 | 0nnndddd | MIDI Time Code Quarter Frame. nnn = Type de message dddd = Valeurs |
| 11110010 | 0lllllll 0mmmmmmm | Pointeur de position de morceau. C'est un registre interne à 14 bits qui contient le nombre de battements MIDI (1 battement = six horloges MIDI) depuis le début du morceau. l est le LSB, m le MSB. |
| 11110011 | 0sssssss | Song Select. The Song Select spécifie la séquence ou le morceau à jouer. |
| 11110100 | | Indéfini. (Réservé) |
| 11110101 | | Indéfini. (Réservé) |
| 11110110 | | Demande de réglage. À la réception d'une demande de réglage, tous les synthétiseurs analogiques doivent accorder leurs oscillateurs. |
| 11110111 | | Fin de la exclusivité. Utilisé pour terminer un vidage exclusif du système (voir ci-dessus). |

**Messages système en temps réel**

| Statut D7..D0 | Octets de données | La description |
|---|---|---|
| 11111000 | | Horloge de chronométrage Envoyé 24 fois par trimestre, lorsque la synchronisation est requise (voir texte). |

| Statut D7..D0 | Octets de données | La description |
|---|---|---|
| 11111001 | | Indéfini. (Réservé) |
| 11111010 | | Début. Lancer la séquence en cours de lecture. (Ce message sera suivi par des horloges de synchronisation). |
| 11111011 | | Continuer. Continuez au point où la séquence a été arrêtée. |
| 11111100 | | Arrêtez. Arrête la séquence en cours. |
| 11111101 | | Indéfini. (Réservé) |
| 11111110 | | Détection active. Ce message est destiné à être envoyé à plusieurs reprises pour indiquer au destinataire qu'une connexion est active. L'utilisation de ce message est facultative. Lorsqu'il est initialement reçu, le récepteur s'attend à recevoir un autre message Active Sensing tous les 300 ms (max) et s'il ne le fait pas, il supposera que la connexion est terminée. À la fin de la communication, le récepteur éteindra toutes les voix et reviendra au fonctionnement normal (détection non active). |
| 11111111 | | Réinitialiser. Réinitialise tous les récepteurs du système au statut de mise sous tension. Ceci doit être utilisé avec parcimonie, de préférence sous contrôle manuel. En particulier, il ne devrait pas être envoyé à la mise sous tension. |

Lire Communication MIDI en ligne: https://riptutorial.com/fr/arduino/topic/9406/communication-midi

# Chapitre 12: Communication série

## Syntaxe

- `Serial.begin(speed)` // Ouvre le port série sur le débit en bauds donné
- `Serial.begin(speed, config)`
- `Serial[1-3].begin(speed)` // **Arduino Mega uniquement!** Lorsque vous écrivez 1-3, cela signifie que vous pouvez choisir entre les numéros 1 à 3 lors du choix du port série.
- `Serial[1-3].begin(speed, config)` // **Arduino Mega seulement!** Lorsque vous écrivez 1-3, cela signifie que vous pouvez choisir entre les numéros 1 à 3 lors du choix du port série.
- `Serial.peek()` // Lit le prochain octet de l'entrée sans le retirer du tampon
- `Serial.available()` // Obtient le nombre d'octets dans le tampon
- `Serial.print(text)` // Écrit le texte sur le port série
- `Serial.println(text)` // Identique à `Serial.print()` mais avec une nouvelle ligne

## Paramètres

| Paramètre | Détails |
|---|---|
| La vitesse | Le taux du port série (généralement 9600) |
| Texte | Le texte à écrire sur le port série (n'importe quel type de données) |
| Bits de données | Nombre de bits de données dans un paquet (de 5 à 8), la valeur par défaut est 8 |
| Parité | Options de parité pour la détection des erreurs: none (par défaut), pair, impair |
| Bits d'arrêt | Nombre de bits d'arrêt dans un paquet: un (par défaut), deux |

## Remarques

L'Arduino Mega dispose de quatre ports série parmi lesquels il est possible de choisir. Ils sont accédés de la manière suivante

```
Serial.begin(9600);
Serial1.begin(38400);
Serial2.begin(19200);
Serial3.begin(4800);
```

Le port série d'un Arduino peut être configuré avec des paramètres supplémentaires. Le paramètre config définit les bits de données, la parité et les bits d'arrêt. Par exemple:

8 bits de données, même parité et 1 bit d'arrêt seraient - `SERIAL_8E1`

6 bits de données, parité impaire et 2 bits d'arrêt seraient - `SERIAL_6O2`

7 bits de données, pas de parité et 1 bit d'arrêt serait - `SERIAL_7N1`

# Examples

## Simple à lire et à écrire

Cet exemple écoute les entrées provenant de la connexion série, puis les répète sur la même connexion.

```
byte incomingBytes;

void setup() {
  Serial.begin(9600); // Opens serial port, sets data rate to 9600 bps.
}

void loop() {
  // Send data only when you receive data.
  if (Serial.available() > 0) {
    // Read the incoming bytes.
    incomingBytes = Serial.read();

    // Echo the data.
    Serial.println(incomingBytes);
  }
}
```

## Filtrage Base64 pour les données d'entrée série

```
String base64="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";

void setup() {

    Serial.begin(9600); // Turn the serial protocol ON
    Serial.println("Start Typing");
}

void loop() {

    if (Serial.available() > 0) { // Check if data has been sent from the user
        char c = Serial.read();   // Gets one byte/Character from serial buffer
        int result = base64.indexOf(c); // Base64 filtering
        if (result>=0)
            Serial.print(c); // Only print Base64 string
    }
}
```

## Gestion des commandes sur série

```
byte incoming;
String inBuffer;

void setup() {
```

```
    Serial.begin(9600); // or whatever baud rate you would like
}

void loop(){
    // setup as non-blocking code
    if(Serial.available() > 0) {
        incoming = Serial.read();

        if(incoming == '\n') {  // newline, carriage return, both, or custom character

            // handle the incoming command
            handle_command();

            // Clear the string for the next command
            inBuffer = "";
        } else{
            // add the character to the buffer
            inBuffer += incoming;
        }
    }

    // since code is non-blocking, execute something else . . . .
}

void handle_command() {
    // expect something like 'pin 3 high'
    String command = inBuffer.substring(0, inBuffer.indexOf(' '));
    String parameters = inBuffer.substring(inBuffer.indexOf(' ') + 1);

    if(command.equalsIgnoreCase('pin')){
        // parse the rest of the information
        int pin = parameters.substring("0, parameters.indexOf(' ')).toInt();
        String state = parameters.substring(parameters.indexOf(' ') + 1);

        if(state.equalsIgnoreCase('high')){
            digitalWrite(pin, HIGH);
        }else if(state.equalsIgnoreCase('low)){
            digitalWrite(pin, LOW);
        }else{
            Serial.println("did not compute");
        }
    } // add code for more commands
}
```

### Communication série avec Python

Si vous avez un Arduino connecté à un ordinateur ou à un Raspberry Pi et que vous souhaitez envoyer des données de l'Arduino au PC, vous pouvez procéder comme suit:

# Arduino:

```
void setup() {
  // Opens serial port, sets data rate to 9600 bps:
  Serial.begin(9600);
}
```

```
void loop() {
  // Sends a line over serial:
  Serial.println("Hello, Python!");
  delay(1000);
}
```

# Python:

```
import serial


ser = serial.Serial('/dev/ttyACM0', 9600)  # Start serial communication
while True:
    data = ser.readline()  # Wait for line from Arduino and read it
    print("Received: '{}'".format(data))  # Print the line to the console
```

Lire Communication série en ligne: https://riptutorial.com/fr/arduino/topic/1674/communication-serie

# Chapitre 13: Communication SPI

## Remarques

# Signaux de sélection de puce

La plupart des esclaves ont une entrée de sélection de puce faible active. Le code approprié pour initialiser et utiliser une broche de sélection de puce est le suivant:

```
#define CSPIN 1 // or whatever else your CS pin is
// init:
pinMode(CSPIN, OUTPUT);
digitalWrite(CSPIN, 1); // deselect

// use:
digitalWrite(CSPIN, 0); // select
... perform data transfer ...
digitalWrite(CSPIN, 1); // deselect
```

La désélection d'un esclave est tout aussi importante que sa sélection, car un esclave peut piloter la ligne MISO pendant sa sélection. Il peut y avoir beaucoup d'esclaves, mais un seul peut conduire MISO. Si un esclave n'est pas désélectionné correctement, deux esclaves ou plus peuvent piloter le MISO, ce qui peut entraîner des courts-circuits entre leurs sorties et endommager les appareils.

# Transactions

Les transactions ont deux objectifs:

- dire au SPI que nous voulons commencer et finir de l'utiliser dans un contexte particulier
- configurer le SPI pour une puce spécifique

La ligne d'horloge a différents états de veille dans les différents modes SPI. Le changement du mode SPI lorsqu'un esclave est sélectionné peut induire en erreur l'esclave. Réglez donc toujours le mode SPI avant de sélectionner un esclave. Le mode SPI peut être défini avec un objet `SPISettings` transmis à `SPI.beginTransaction` :

```
SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
digitalWrite(CSPIN, 0);
... perform data transfer ...
digitalWrite(CSPIN, 1);
SPI.endTransaction();
```

`SPISettings` peuvent également être stockés ailleurs:

```
SPISettings mySettings(1000000, MSBFIRST, SPI_MODE0);
```

```
SPI.beginTransaction(mySettings);
```

Si une autre partie du code tente d'utiliser le SPI entre une paire d'appels à `beginTransaction()` et `endTransaction()` , une erreur peut survenir - la manière dont cela se produit dépend de l'implémentation.

Voir aussi la référence Arduino: SPISettings

# Utilisation du SPI dans les routines du service d'interruption

Si le SPI doit être utilisé dans un ISR, aucune autre transaction ne peut avoir lieu en même temps. La bibliothèque SPI fournit `usingInterrupt(interrupt_number)` pour faciliter cela. Cela fonctionne en désactivant l'interruption donnée chaque fois que `beginTransaction()` est appelée, de sorte que l'interruption ne peut pas se déclencher entre cette paire d'appels à `beginTransaction()` et `endTransaction()` .

Voir aussi la référence Arduino: SPI: usingInterrupt

## Examples

### Notions de base: initialiser le SPI et une broche de sélection de puce, et effectuer un transfert d'un octet

```
#include <SPI.h>
#define CSPIN 1

void setup() {
  pinMode(CSPIN, OUTPUT); // init chip select pin as an output
  digitalWrite(CSPIN, 1); // most slaves interpret a high level on CS as "deasserted"

  SPI.begin();

  SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
  digitalWrite(CSPIN, 0);

  unsigned char sent = 0x01;
  unsigned char received = SPI.transfer(sent);
  // more data could be transferred here

  digitalWrite(CSPIN, 1);
  SPI.endTransaction();

  SPI.end();
}

void loop() {
  // we don't need loop code in this example.
}
```

Cet exemple:

- initialise correctement et utilise une broche de sélection de puce (voir remarques)
- utilise correctement une transaction SPI (voir remarques)
- utilise uniquement le SPI pour transférer un seul octet. Il existe également une méthode de transfert de tableaux, qui n'est pas utilisée ici.

Lire Communication SPI en ligne: https://riptutorial.com/fr/arduino/topic/4919/communication-spi

# Chapitre 14: Entrées analogiques

## Syntaxe

- `analogRead(pin)` // Lecture depuis la broche donnée.

## Remarques

```
Serial.println(val)
```

Pour obtenir de l'aide sur la communication série, voir: Communication série

## Examples

### Imprimer une valeur analogique

```
int val = 0;    // variable used to store the value
                // coming from the sensor

void setup() {
  Serial.begin(9600); //Begin serializer to print out value

  // Note: Analogue pins are
  // automatically set as inputs
}

void loop() {

  val = analogRead(0); // read the value from
                       // the sensor connected to A0.

  Serial.println(val); //Prints the value coming in from the analog sensor

  delay(10); // stop the program for
             // some time
}
```

### Obtenir la tension de la broche analogique

Les broches analogiques peuvent être utilisées pour lire des tensions utiles pour la surveillance de la batterie ou pour l'interfaçage avec des périphériques analogiques. Par défaut, la broche AREF sera la même que la tension de fonctionnement de l'arduino, mais peut être réglée sur d'autres valeurs en externe. Si la tension à lire est supérieure à la tension d'entrée, un diviseur potentiel sera nécessaire pour abaisser la tension analogique.

```
#define analogPin 14    //A0 (uno)
#define AREFValue 5        //Standard for 5V Arduinos
#define ADCResolution 1023    //Standard for a 10bit ADC
```

```
int ADCValue = 0;
float voltage = 0;

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    readADC();
    Serial.print(voltage); Serial.println("V");
}

void readADC()
{
    ADCValue = analogRead(analogPin);
    float = ( ( (float)ADCValue/ADCRange ) * AREFValue );    //Convert the ADC value to a
float, devide by the ADC resolution and multiply by the AREF voltage
}
```

Lire Entrées analogiques en ligne: https://riptutorial.com/fr/arduino/topic/2382/entrees-analogiques

# Chapitre 15: Entrées Numériques

## Syntaxe

- `pinMode(pin, pinMode)` // Définit la broche sur le mode défini.
- `digitalRead(pin);` // Lit la valeur d'une broche numérique spécifiée,

## Paramètres

| Paramètre | Détails |
|---|---|
| Pinmode | Devrait être réglé sur `INPUT` ou `INPUT_PULLUP` |

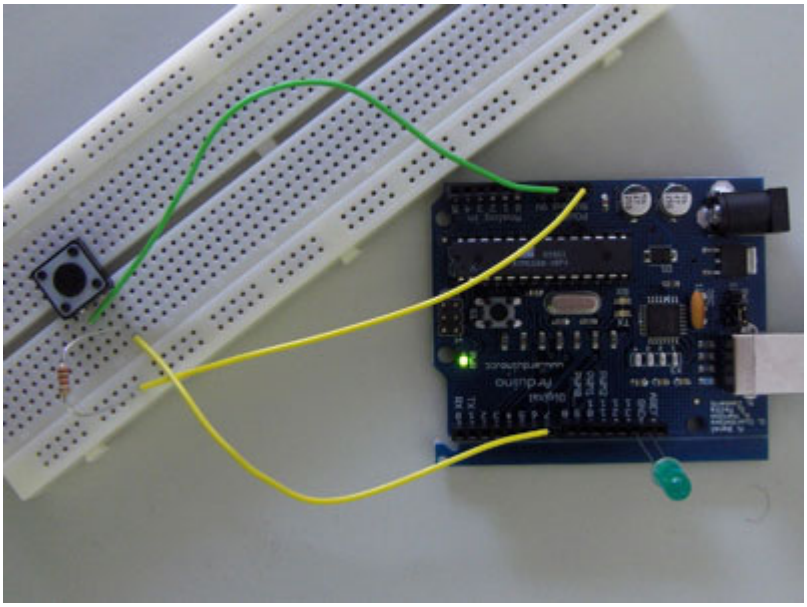
## Remarques

Si la broche d'entrée n'est pas tirée LOW ou HIGH, la valeur flottera. Autrement dit, ce ne sera pas clairement un 1 ou un 0, mais quelque part entre les deux. Pour une entrée numérique, une résistance de pullup ou de pulldown est une nécessité.

## Examples

### Lecture bouton

Voici un exemple de base sur la manière de câbler et d'allumer / éteindre une LED lorsque le bouton-poussoir est enfoncé.



```
/* Basic Digital Read
 * ------------------
```

```
 *
 * turns on and off a light emitting diode(LED) connected to digital
 * pin 13, when pressing a pushbutton attached to pin 7. It illustrates the
 * concept of Active-Low, which consists in connecting buttons using a
 * 1K to 10K pull-up resistor.
 *
 * Created 1 December 2005
 * copyleft 2005 DojoDave <http://www.0j0.org>
 * http://arduino.berlios.de
 *
 */

int ledPin = 13; // choose the pin for the LED
int inPin = 7;   // choose the input pin (for a pushbutton)
int val = 0;     // variable for reading the pin status

void setup() {
  pinMode(ledPin, OUTPUT);  // declare LED as output
  pinMode(inPin, INPUT);    // declare pushbutton as input
}

void loop(){
  val = digitalRead(inPin);  // read input value
  if (val == HIGH) {         // check if the input is HIGH (button released)
    digitalWrite(ledPin, LOW);  // turn LED OFF
  } else {
    digitalWrite(ledPin, HIGH);  // turn LED ON
  }
}
```

Exemple tiré d' Arduino.cc .

Lire Entrées Numériques en ligne: https://riptutorial.com/fr/arduino/topic/1662/entrees-numeriques

# Chapitre 16: Gestion du temps

## Syntaxe

- non signé long millis ()
- non signé long micros ()
- délai d'annulation (millisecondes longues non signées)
- délai videMicrosecondes (microsecondes longues non signées)
- Voir l'en-tête elapsedMillis pour les constructeurs et les opérateurs de cette classe. En bref:
    - elapsedMillis elapsedMillisObject; *crée un objet pour suivre le temps écoulé depuis sa création ou depuis un autre point défini explicitement dans le temps*
    - elapsedMillisObject = 0; *réinitialiser le temps suivi par l'objet à "depuis maintenant"*
    - unsigned long deltaT = elapsedMillisObject; *nous permet de regarder le temps suivi*
    - elapsedMillisObject + = et - = *ceux-ci fonctionnent comme prévu*

## Remarques

## Code bloquant ou non bloquant

Pour des esquisses très simples, écrire du code de blocage à l'aide de `delay()` et `delayMicroseconds()` peut être approprié. Lorsque les choses deviennent plus complexes, l'utilisation de ces fonctions peut présenter certains inconvénients. Certains d'entre eux sont:

- Perte de temps CPU: des croquis plus complexes peuvent nécessiter le CPU pour autre chose en attendant la fin de la période de clignotement des voyants.
- Retards inattendus: lorsque `delay()` est appelé dans des sous-routines qui ne sont pas appelées de manière évidente, par exemple dans les bibliothèques que vous incluez.
- les événements manquants qui se produisent pendant le délai et qui ne sont pas traités par un gestionnaire d'interruption, par exemple une pression sur le bouton d'interrogation: un bouton peut être pressé pendant 100 ms, mais cela peut être `delay(500)` .

## Détails d'implémentation

`millis()` repose généralement sur une minuterie matérielle dont la vitesse est bien supérieure à 1 kHz. Lorsque `millis()` est appelée, l'implémentation retourne une valeur, mais vous ne savez pas quel âge elle a. Il est possible que la milliseconde "courante" vienne juste de démarrer ou qu'elle se termine juste après cet appel de fonction. Cela signifie que, lors du calcul de la différence entre deux résultats à partir de `millis()` , vous pouvez être désactivé par presque zéro et presque une milliseconde. Utilisez `micros()` si une plus grande précision est nécessaire.

En `elapsedMillis` code source de `elapsedMillis` que `millis()` interne pour comparer deux points dans le temps, ce qui nuit également à cet effet. Encore une fois, il existe une alternative `elapsedMicros` pour une plus grande précision, à partir de la même bibliothèque.

# Examples

## blinky blinky avec delay ()

Une des manières les plus simples de faire clignoter une LED est: allumez-la, attendez un peu, éteignez-la, attendez encore et répétez sans cesse:

```
// set constants for blinking the built-in LED at 1 Hz
#define OUTPIN LED_BUILTIN
#define PERIOD 500

void setup()
{
  pinMode(OUTPIN, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  digitalWrite(OUTPIN, HIGH);   // sets the pin on
  delayMicroseconds(PERIOD);        // pauses for 500 miliseconds
  digitalWrite(OUTPIN, LOW);    // sets the pin off
  delayMicroseconds(PERIOD);        // pauses for 500 milliseconds

  // doing other time-consuming stuff here will skew the blinking
}
```

Cependant, attendre comme dans l'exemple ci-dessus gaspille les cycles du processeur, car il ne fait que rester en boucle pendant un certain temps. C'est ce que les méthodes non bloquantes, à l'aide de `millis()` ou `elapsedMillis` , font mieux - dans le sens où elles ne brûlent pas autant de fonctionnalités du matériel.

## Blinky non bloquant avec la bibliothèque elapsedMillis (et la classe)

La bibliothèque elapsedMillis fournit une classe avec le même nom qui garde la trace du temps écoulé depuis sa création ou sa valeur:

```
#include <elapsedMillis.h>

#define OUTPIN LED_BUILTIN
#define PERIOD 500

elapsedMillis ledTime;

bool ledState = false;

void setup()
{
  // initialize the digital pin as an output.
  pinMode(OUTPIN, OUTPUT);
```

```
}

void loop()
{
    if (ledTime >= PERIOD)
    {
        ledState = !ledState;
        digitalWrite(OUTPIN, ledState);
        ledTime = 0;
    }
    // do other stuff here
}
```

Vous pouvez voir dans l'exemple que l'objet `ledTime` est assigné à zéro lorsque la broche du voyant a été basculée. Cela n'est peut-être pas surprenant à première vue, mais cela se produit si des choses plus chronophages se produisent:

Considérons une situation où la comparaison entre `ledTime` et `PERIOD` se fait après 750 millisecondes. Le réglage de `ledTime` à zéro signifie que toutes les opérations de bascule suivantes seront 250 ms en retard. Si, au contraire, `PERIOD` a été soustrait de `ledTime` , la LED verrait une courte période, puis continuer à clignoter comme si rien ne se passait.

## Blinky non bloquant avec le millis ()

Ceci est très proche d' un exemple de la documentation arduino :

```
// set constants for blinking the built-in LED at 1 Hz
#define OUTPIN LED_BUILTIN
#define PERIOD 500  // this is in milliseconds

int ledState = LOW;

// millis() returns an unsigned long so we'll use that to keep track of time
unsigned long lastTime = 0;

void setup() {
  // set the digital pin as output:
  pinMode(OUTPIN, OUTPUT);
}

void loop() {
  unsigned long now = millis();
  if (now - lastTime >= PERIOD) // this will be true every PERIOD milliseconds
  {
    lastTime = now;
    if (ledState == LOW)
    {
      ledState = HIGH;
    }
    else
    {
      ledState = LOW;
    }
    digitalWrite(OUTPIN, ledState);
  }

  // now there's lots of time to do other stuff here
```

```
}
```

Utiliser `millis()` de cette façon - pour chronométrer les opérations de manière non bloquante - est quelque chose qui est nécessaire assez fréquemment, alors envisagez d'utiliser la bibliothèque `elapsedMillis` pour cela.

## Mesurer combien de temps quelque chose a pris, en utilisant Millis et ElapsedMicros

```
#include <elapsedMillis.h>

void setup() {
  Serial.begin(115200);
  elapsedMillis msTimer;
  elapsedMicros usTimer;

  long int dt = 500;
  delay(dt);

  long int us = usTimer;
  long int ms = msTimer;

  Serial.print("delay(");Serial.print(dt);Serial.println(") took");
  Serial.print(us);Serial.println(" us, or");
  Serial.print(ms);Serial.println(" ms");
}

void loop() {
}
```

Dans cet exemple, un objet `elapsedMillis` et un objet `elapsedMicros` sont utilisés pour mesurer combien de temps quelque chose a pris, en les créant juste avant que l'expression que nous voulons chronométrer soit exécutée, et en obtenant ensuite leurs valeurs. Ils afficheront des résultats légèrement différents, mais le résultat en millisecondes ne sera pas désactivé par plus d'une milliseconde.

## Plus d'une tâche sans délai ()

Si vous avez plus d'une tâche à exécuter à plusieurs reprises dans des intervalles différents, utilisez cet exemple comme point de départ:

```
unsigned long intervals[] = {250,2000}; //this defines the interval for each task in
milliseconds
unsigned long last[] = {0,0};           //this records the last executed time for each task

void setup() {
  pinMode(LED_BUILTIN, OUTPUT); //set the built-it led pin as output
  Serial.begin(115200);         //initialize serial
}

void loop() {
  unsigned long now = millis();
  if(now-last[0]>=intervals[0]){ last[0]=now; firstTask(); }
  if(now-last[1]>=intervals[1]){ last[1]=now; secondTask(); }
```

```
  //do other things here
}

void firstTask(){
  //let's toggle the built-in led
  digitalWrite(LED_BUILTIN, digitalRead(LED_BUILTIN)?0:1);
}

void secondTask(){
  //say hello
  Serial.println("hello from secondTask()");
}
```

Pour ajouter une autre tâche à exécuter toutes les 15 secondes, étendez les `intervals` variables et `last` :

```
unsigned long intervals[] = {250,2000,15000};
unsigned long last[] = {0,0,0};
```

Ajoutez ensuite une instruction `if` pour exécuter la nouvelle tâche. Dans cet exemple, je l'ai nommé `thirdTask` .

```
if(now-last[2]>=intervals[2]){ last[2]=now; thirdTask(); }
```

Enfin déclarez la fonction:

```
void thirdTask(){
  //your code here
}
```

Lire Gestion du temps en ligne: https://riptutorial.com/fr/arduino/topic/4852/gestion-du-temps

# Chapitre 17: Les fonctions

## Remarques

En dehors du C / C ++ ordinaire, l'IDE Arduino permet d'appeler une fonction avant sa définition.

Dans les fichiers .cpp, vous devez définir la fonction ou du moins déclarer le prototype de fonction avant de pouvoir l'utiliser.
Dans un fichier .ino, l'IDE Arduino crée un tel prototype dans les coulisses.

Arduino - déclaration de fonction - officielle

## Examples

### Créer une fonction simple

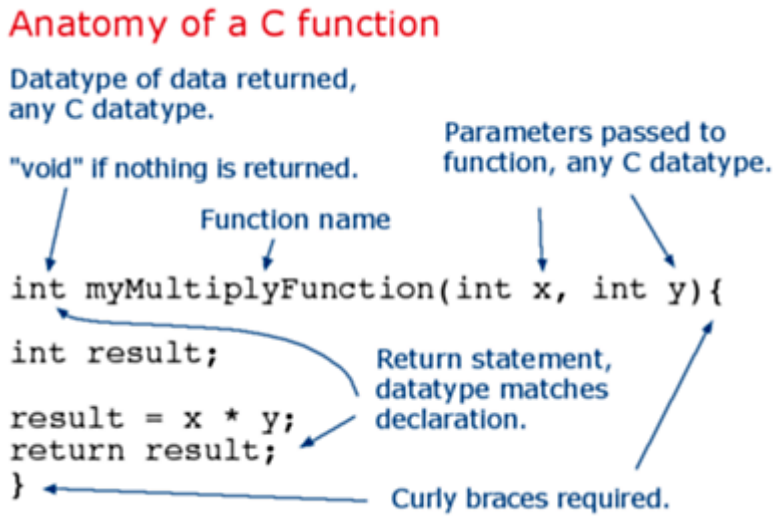
```
int squareNum (int a) {
    return a*a;
}
```

**`int`** : type de retour

**`squareNum`** : nom de la fonction

**`int a`** : type et nom du paramètre

**`return a*a`** : retourne une valeur (même type que le type de retour défini au début)




### Appeler une fonction

Si vous avez une fonction déclarée, vous pouvez l'appeler n'importe où dans le code. Voici un exemple d'appel d'une fonction:

```
void setup(){
  Serial.begin(9600);
}

void loop() {
  int i = 2;

  int k = squareNum(i); // k now contains 4
  Serial.println(k);
  delay(500);
}

int squareNum(int a) {
    return a*a;
}
```

Lire Les fonctions en ligne: https://riptutorial.com/fr/arduino/topic/2380/les-fonctions

# Chapitre 18: Les interruptions

## Syntaxe

- digitalPinToInterrupt (broche); // convertit un identifiant de broche en identifiant d'interruption, à utiliser avec `attachInterrupt()` et `detachInterrupt()` .
- attachInterrupt (digitalPinToInterrupt (pin), ISR, mode); // conseillé
- attachInterrupt (interruption, ISR, mode); // non recommandé
- detachInterrupt (digitalPinToInterrupt (pin));
- detachInterrupt (interruption);
- noInterrupts (); // désactive les interruptions
- interruptions (); // réactiver les interruptions après l' `noInterrupts()` de `noInterrupts()` .

## Paramètres

| Paramètre | Remarques |
|---|---|
| interrompre | Identifiant de l'interruption. Ne pas être confondu avec le numéro d'identification. |
| ISR | Routine de service d'interruption. C'est la méthode qui sera exécutée lorsque l'interruption se produira. |
| mode | Qu'est-ce qui devrait déclencher l'interruption? Un de bas, changer, se lever ou tomber. Les commissions sont également autorisées. |

## Remarques

Les routines de service d'interruption (ISR) doivent être aussi courtes que possible, car elles mettent en pause l'exécution du programme principal et peuvent donc vider le code en fonction du temps. Généralement, cela signifie que dans l'ISR, vous définissez un drapeau et sortez, et dans la boucle du programme principal, vous vérifiez le drapeau et faites ce que ce drapeau est censé faire.

Vous ne pouvez pas utiliser `delay()` ou `millis()` dans un ISR car ces méthodes elles-mêmes reposent sur des interruptions.

## Examples

### Interruption sur le bouton presse

Cet exemple utilise un bouton-poussoir (commutateur tactile) connecté à la broche numérique 2 et à la masse, en utilisant une résistance de rappel interne pour que la broche 2 soit haute lorsque le bouton n'est pas enfoncé.

```
const int LED_PIN = 13;
const int INTERRUPT_PIN = 2;
volatile bool ledState = LOW;

void setup() {
    pinMode(LED_PIN, OUTPUT);
    pinMode(INTERRUPT_PIN, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), myISR, FALLING); // trigger when
button pressed, but not when released.
}

void loop() {
    digitalWrite(LED_PIN, ledState);
}

void myISR() {
    ledState = !ledState;
    // note: LOW == false == 0, HIGH == true == 1, so inverting the boolean is the same as
switching between LOW and HIGH.
}
```

Un exemple avec cet exemple simple est que les boutons-poussoirs ont tendance à rebondir, ce qui signifie que le circuit s’ouvre et se ferme plus d’une fois avant de s’établir dans l’état final fermé ou ouvert. Cet exemple ne prend pas cela en compte. Par conséquent, il suffit parfois d'appuyer sur le bouton pour basculer le voyant plusieurs fois, au lieu d'une fois.

Lire Les interruptions en ligne: https://riptutorial.com/fr/arduino/topic/2913/les-interruptions

# Chapitre 19: Nombres aléatoires

## Syntaxe

- random (max) // Retourne un nombre (long) pseudo-aléatoire compris entre 0 (inclus) et max (exclusif)
- random (min, max) // Retourne un nombre pseudo-aléatoire (long) entre min (inclus) et max (exclusif)
- randomSeed (seed) // Initialise le générateur de nombres pseudo-aléatoires, le faisant démarrer à un point spécifié de sa séquence.

## Paramètres

| Paramètre | Détails |
|---|---|
| min | La valeur minimale possible (incluse) à générer par la fonction `random()` . |
| max | La valeur maximale possible (exclusive) à générer par la fonction `random()` . |
| la graine | La graine qui sera utilisée pour mélanger la fonction `random()` . |

## Remarques

Si `randomSeed()` est appelée avec une valeur fixe (par exemple, `randomSeed(5)` ), la séquence de nombres aléatoires générée par l'esquisse sera répétée à chaque exécution. Dans la plupart des cas, une graine aléatoire est préférée, qui peut être obtenue en lisant une broche analogique non connectée.

## Examples

### Générer un nombre aléatoire

La fonction `random()` peut être utilisée pour générer des nombres pseudo-aléatoires:

```
void setup() {
    Serial.begin(9600);
}

void loop() {
    long randomNumber = random(500);  // Generate a random number between 0 and 499
    Serial.println(randomNumber);

    randomNumber = random(100, 1000); // Generate a random number between 100 and 999
    Serial.println(randomNumber);
```

```
    delay(100);
}
```

## Mettre une graine

S'il est important qu'une séquence de nombres générée par `random()` diffère, il est `randomSeed()` de spécifier une graine avec `randomSeed()` :

```
void setup() {
    Serial.begin(9600);

    // If analog pin 0 is left unconnected, analogRead(0) will produce a
    // different random number each time the sketch is run.
    randomSeed(analogRead(0));
}

void loop() {
    long randomNumber = random(500); // Generate a random number between 0 and 499
    Serial.println(randomNumber);

    delay(100);
}
```

Lire Nombres aléatoires en ligne: https://riptutorial.com/fr/arduino/topic/2238/nombres-aleatoires

# Chapitre 20: PWM - Modulation de la largeur d'impulsion

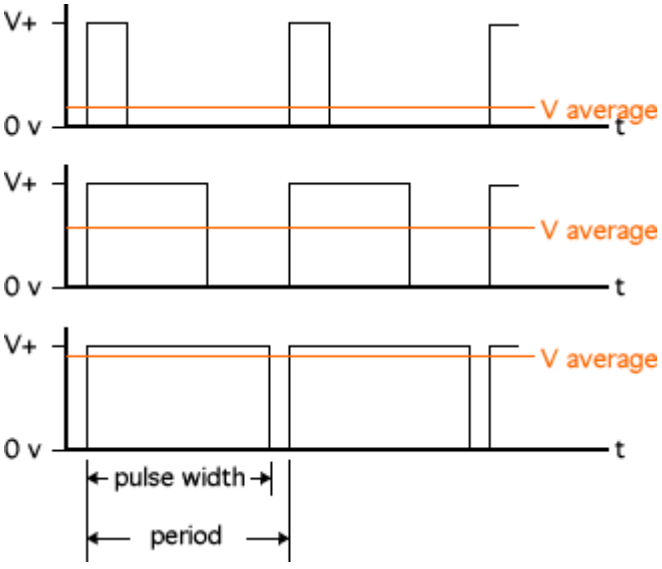
## Examples

### Contrôler un moteur à courant continu via le port série en utilisant PWM

Dans cet exemple, nous visons à accomplir l'une des tâches les plus courantes: *un petit moteur à courant continu est installé, comment puis-je utiliser mon Arduino pour le contrôler?* Facile, avec communication PWM et série, en utilisant la fonction `analogWrite()` et la bibliothèque `Serial` .

# Les bases

La modulation de largeur d'impulsion ou PWM en abrégé est une technique permettant d'imiter des signaux analogiques en utilisant une sortie numérique. Comment cela marche-t-il? En utilisant un train d'impulsions dont la relation D (rapport cyclique) entre le temps à haut niveau (numérique 1, généralement 5V) et le temps à bas niveau (numérique 0, 0V) dans chaque période peut être modifiée pour produire une tension moyenne entre ces deux niveaux:




En utilisant la fonction `analogWrite(pin,value)` Arduino `analogWrite(pin,value)` nous pouvons faire varier la `value` du rapport cyclique de la sortie de la `pin` . Notez que la `pin` doit être mise en mode de sortie et que la `value` doit être comprise entre 0 (0V) et 255 (5V). Toute valeur intermédiaire simulera une sortie analogique intermédiaire proportionnelle.

Cependant, la fonction des signaux analogiques est généralement liée au contrôle de systèmes mécaniques qui nécessitent plus de tension et de courant que la carte Arduino seule. Dans cet exemple, nous allons apprendre à amplifier les capacités PWM d'Arduino.

Pour cela, une diode MOSFET est utilisée. Essentiellement, cette diode agit comme un

commutateur. Il permet ou interrompt le flux électrique entre ses bornes de *source* et de *drain* . Mais au lieu d'un commutateur mécanique, il dispose d'un troisième terminal appelé *gate* . Un très petit courant (<1mA) "ouvrira" cette porte et permettra au courant de circuler. Ceci est très pratique, car nous pouvons envoyer la sortie PWM d'Arduino à cette porte, créant ainsi *un autre* train d'impulsions PWM avec le même rapport cyclique à travers le MOSFET, ce qui permet des tensions et des courants qui pourraient détruire l'Arduino.

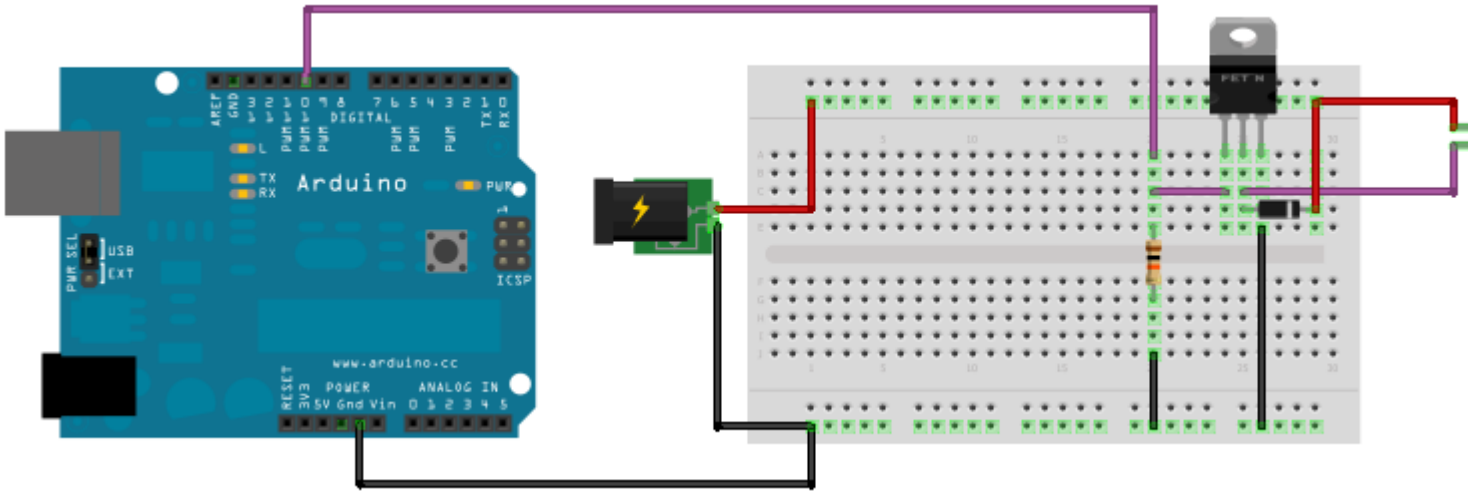
# Bill of materials: de quoi avez-vous besoin pour construire cet exemple?

- Diode MOSFET: par exemple, le populaire BUZ11
- Diode de protection pour le moteur: Schottky SB320
- Résistance: rien 10K ~ 1M Ohm
- Moteur: Un petit moteur typique (un moteur typique peut être 12V)
- Une source d'alimentation compatible avec le moteur que vous avez sélectionné
- Une planche à pain
- Câbles colorés!
- Un Arduino, mais tu le savais déjà.

# La construction

Mettez tout ensemble! Alimentez les rails de la maquette et placez-y la diode MOSFET. Connectez le moteur entre le rail positif et le drain MOSFET. Connecter la diode de protection de la même manière: entre le drain MOSFET et le rail positif. Connectez la source du MOSFET au rail de terre commun. Enfin, connectez la broche PWM (nous utilisons la broche 10 dans cet exemple) à la grille du MOSFET et également à la terre commune via la résistance (nous avons besoin d'un courant très faible!).

Voici un exemple de l'apparence de cette construction. Si vous préférez un schéma, en voici un.



# Le code

Maintenant, nous pouvons connecter l'Arduino à un ordinateur, télécharger le code et contrôler le moteur en envoyant des valeurs via la communication série. Rappelons que ces valeurs doivent être des entiers compris entre 0 et 255. Le code réel de cet exemple est très simple. Une explication est fournie dans chaque ligne.

```
int in = 0;                    // Variable to store the desired value
byte pinOut = 10;              // PWM output pin

void setup() {                 // This executes once
  Serial.begin(9600);              // Initialize serial port
  pinMode(pinOut, OUTPUT);         // Prepare output pin
}

void loop() {                  // This loops continuously
  if(Serial.available()){          // Check if there's data
    in = Serial.read();            // Read said data into the variable "in"
    analogWrite(pinOut, in);       // Pass the value of "in" to the pin
  }
}
```

Et c'est tout! Vous pouvez désormais utiliser les fonctionnalités PWM d'Arduino pour contrôler les applications qui nécessitent des signaux analogiques, même lorsque les besoins en énergie dépassent les limites de la carte.

## PWM avec un TLC5940

Le TLC5940 est un élément pratique à avoir lorsque vous manquez de ports PWM sur l'Arduino. Il dispose de 16 canaux, chacun individuellement contrôlable avec 12 bits de résolution (0-4095). Une bibliothèque existante est disponible à l' adresse http://playground.arduino.cc/Learning/TLC5940 . Il est utile pour contrôler plusieurs servos ou LED RVB. N'oubliez pas que les LED doivent être des anodes communes pour fonctionner. De plus, les puces peuvent être chaînées en série, permettant encore plus de ports PWM.

Exemple:

```
// Include the library
#include <Tlc5940.h>

void setup() {
    // Initialize
    Tlc.init();
    Tlc.clear();
}

unsigned int level = 0;
void loop() {
    // Set all 16 outputs to same value
    for (int i = 0; i < 16; i++) {
        Tlc.set(i, level);
    }
```

```
    level = (level + 1) % 4096;
    // Tell the library to send the values to the chip
    Tlc.update();
    delay(10);
}
```

Lire PWM - Modulation de la largeur d'impulsion en ligne:
https://riptutorial.com/fr/arduino/topic/1658/pwm---modulation-de-la-largeur-d-impulsion

# Chapitre 21: Servo

## Introduction

Un Servo est un système fermé contenant un moteur et des circuits de support. L'arbre d'un servo peut être tourné à un angle fixe dans un arc en utilisant un signal de commande. Si le signal de contrôle est maintenu, le servo conservera son angle. Les servos peuvent facilement être contrôlés avec la bibliothèque Arduino `Servo.h` .

## Syntaxe

- #include <Servo.h> // Inclure la bibliothèque de servos
- Servo.attach (pin) // Fixe le servo sur la broche. Renvoie un objet Servo
- Servo.write (degrés) // Degrés pour passer à (0 - 180)
- Servo.read () // Obtient la rotation actuelle du servo

## Examples

### Déplacement du servo dans les deux sens

```
#include <Servo.h>

Servo srv;

void setup() {
  srv.attach(9); // Attach to the servo on pin 9

}
```

Pour utiliser un servo, vous devez d'abord appeler la fonction `attach()` . Il commence à générer un signal PWM contrôlant un servo sur une broche spécifiée. Sur les cartes autres qu'Arduino Mega, l'utilisation de la bibliothèque Servo désactive la fonctionnalité analogWrite () (PWM) sur les broches 9 et 10, qu'il y ait ou non un Servo sur ces broches.

```
void loop() {
  Servo.write(90); // Move the servo to 90 degrees
  delay(1000); // Wait for it to move to it's new position
  Servo.write(0); // Move the servo to 0 degrees
  delay(1000); // Wait for it to move to it's new position
}
```

Notez que vous n'êtes pas assuré que le servo a atteint la position souhaitée, ni que vous pouvez le vérifier depuis le programme.

Lire Servo en ligne: https://riptutorial.com/fr/arduino/topic/4920/servo

# Chapitre 22: Sortie audio

## Paramètres

| Paramètre | Détails |
|---|---|
| orateur | Devrait être une sortie vers un haut-parleur analogique |

## Examples

### Sorties de note de base

```
#define NOTE_C4  262 //From pitches.h file defined in [Arduino Tone Tutorial][1]

int Key = 2;
int KeyVal = 0;

byte speaker = 12;

void setup()
{
  pinMode(Key, INPUT);  //Declare our key (button) as input
  pinMode(speaker, OUTPUT);
}

void loop()
{
  KeyVal = digitalRead(Key);
  if (KeyVal == HIGH) {
    tone(speaker, NOTE_C4); //Sends middle C tone out through analog speaker
  } else {
    noTone(speaker); //Ceases tone emitting from analog speaker
  }

  delay(100);
}
```

[1]: https://www.arduino.cc/en/Tutorial/toneMelody

Lire Sortie audio en ligne: https://riptutorial.com/fr/arduino/topic/2384/sortie-audio

# Chapitre 23: Sortie numérique

## Syntaxe

- `digitalWrite(pin, value)`

## Examples

### Ecrire à l'épinglette

```
int ledPin = 13;                 // LED connected to digital pin 13

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

Exemple à Arduino.cc .

Lire Sortie numérique en ligne: https://riptutorial.com/fr/arduino/topic/2477/sortie-numerique

# Chapitre 24: Stockage de données

## Examples

### cardInfo

```
/*
 SD card test

 This example shows how use the utility libraries on which the'
 SD library is based in order to get info about your SD card.
 Very useful for testing a card when you're not sure whether its working or not.

 The circuit:
  * SD card attached to SPI bus as follows:
 ** MOSI - pin 11 on Arduino Uno/Duemilanove/Diecimila
 ** MISO - pin 12 on Arduino Uno/Duemilanove/Diecimila
 ** CLK -  pin 13 on Arduino Uno/Duemilanove/Diecimila
 ** CS -   depends on your SD card shield or module.
           Pin 4 used here for consistency with other Arduino examples

 created  28 Mar 2011
 by Limor Fried
 modified 9 Apr 2012
 by Tom Igoe
 */

// include the SD library:
#include <SPI.h>
#include <SD.h>

// set up variables using the SD utility library functions:
Sd2Card card;
SdVolume volume;
SdFile root;

// change this to match your SD shield or module;
// Arduino Ethernet shield: pin 4
// Adafruit SD shields and modules: pin 10
// Sparkfun SD shield: pin 8
const int chipSelect = 4;

void setup()
{
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }

  Serial.print("\nInitializing SD card...");

  // we'll use the initialization code from the utility libraries
  // since we're just testing if the card is working!
  if (!card.init(SPI_HALF_SPEED, chipSelect)) {
    Serial.println("initialization failed. Things to check:");
    Serial.println("* is a card inserted?");
```

```
    Serial.println("* is your wiring correct?");
    Serial.println("* did you change the chipSelect pin to match your shield or module?");
    return;
  } else {
    Serial.println("Wiring is correct and a card is present.");
  }

  // print the type of card
  Serial.print("\nCard type: ");
  switch (card.type()) {
    case SD_CARD_TYPE_SD1:
      Serial.println("SD1");
      break;
    case SD_CARD_TYPE_SD2:
      Serial.println("SD2");
      break;
    case SD_CARD_TYPE_SDHC:
      Serial.println("SDHC");
      break;
    default:
      Serial.println("Unknown");
  }

  // Now we will try to open the 'volume'/'partition' - it should be FAT16 or FAT32
  if (!volume.init(card)) {
    Serial.println("Could not find FAT16/FAT32 partition.\nMake sure you've formatted the
card");
    return;
  }

  // print the type and size of the first FAT-type volume
  uint32_t volumesize;
  Serial.print("\nVolume type is FAT");
  Serial.println(volume.fatType(), DEC);
  Serial.println();

  volumesize = volume.blocksPerCluster();    // clusters are collections of blocks
  volumesize *= volume.clusterCount();       // we'll have a lot of clusters
  volumesize *= 512;                            // SD card blocks are always 512 bytes
  Serial.print("Volume size (bytes): ");
  Serial.println(volumesize);
  Serial.print("Volume size (Kbytes): ");
  volumesize /= 1024;
  Serial.println(volumesize);
  Serial.print("Volume size (Mbytes): ");
  volumesize /= 1024;
  Serial.println(volumesize);


  Serial.println("\nFiles found on the card (name, date and size in bytes): ");
  root.openRoot(volume);

  // list all files in the card with date and size
  root.ls(LS_R | LS_DATE | LS_SIZE);
}

void loop(void) {

}
```

## Enregistreur de carte SD

```
/*
  SD card datalogger

 This example shows how to log data from three analog sensors
 to an SD card using the SD library.

 The circuit:
 * analog sensors on analog ins 0, 1, and 2
 * SD card attached to SPI bus as follows:
 ** MOSI - pin 11
 ** MISO - pin 12
 ** CLK - pin 13
 ** CS - pin 4

 created  24 Nov 2010
 modified 9 Apr 2012
 by Tom Igoe

 This example code is in the public domain.

 */

#include <SPI.h>
#include <SD.h>

const int chipSelect = 4;

void setup()
{
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }

  Serial.print("Initializing SD card...");

  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    return;
  }
  Serial.println("card initialized.");
}

void loop()
{
  // make a string for assembling the data to log:
  String dataString = "";

  // read three sensors and append to the string:
  for (int analogPin = 0; analogPin < 3; analogPin++) {
    int sensor = analogRead(analogPin);
    dataString += String(sensor);
    if (analogPin < 2) {
      dataString += ",";
    }
```

```
  }

  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.
  File dataFile = SD.open("datalog.txt", FILE_WRITE);

  // if the file is available, write to it:
  if (dataFile) {
    dataFile.println(dataString);
    dataFile.close();
    // print to the serial port too:
    Serial.println(dataString);
  }
  // if the file isn't open, pop up an error:
  else {
    Serial.println("error opening datalog.txt");
  }
}
```

## Dump de fichier de carte SD

```
/*
  SD card file dump

 This example shows how to read a file from the SD card using the
 SD library and send it over the serial port.

 The circuit:
 * SD card attached to SPI bus as follows:
 ** MOSI - pin 11
 ** MISO - pin 12
 ** CLK - pin 13
 ** CS - pin 4

 created  22 December 2010
 by Limor Fried
 modified 9 Apr 2012
 by Tom Igoe

 This example code is in the public domain.

 */

#include <SPI.h>
#include <SD.h>

const int chipSelect = 4;

void setup()
{
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }


  Serial.print("Initializing SD card...");
```

```
  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    return;
  }
  Serial.println("card initialized.");

  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.
  File dataFile = SD.open("datalog.txt");

  // if the file is available, write to it:
  if (dataFile) {
    while (dataFile.available()) {
      Serial.write(dataFile.read());
    }
    dataFile.close();
  }
  // if the file isn't open, pop up an error:
  else {
    Serial.println("error opening datalog.txt");
  }
}

void loop()
{

}
```

## Exemple de fichier de base de carte SD

```
/*
  SD card basic file example

 This example shows how to create and destroy an SD card file
 The circuit:
 * SD card attached to SPI bus as follows:
 ** MOSI - pin 11
 ** MISO - pin 12
 ** CLK - pin 13
 ** CS - pin 4

 created   Nov 2010
 by David A. Mellis
 modified 9 Apr 2012
 by Tom Igoe

 This example code is in the public domain.

 */
#include <SPI.h>
#include <SD.h>

File myFile;

void setup()
{
  // Open serial communications and wait for port to open:
```

```
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }

  Serial.print("Initializing SD card...");

  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  if (SD.exists("example.txt")) {
    Serial.println("example.txt exists.");
  }
  else {
    Serial.println("example.txt doesn't exist.");
  }

  // open a new file and immediately close it:
  Serial.println("Creating example.txt...");
  myFile = SD.open("example.txt", FILE_WRITE);
  myFile.close();

  // Check to see if the file exists:
  if (SD.exists("example.txt")) {
    Serial.println("example.txt exists.");
  }
  else {
    Serial.println("example.txt doesn't exist.");
  }

  // delete the file:
  Serial.println("Removing example.txt...");
  SD.remove("example.txt");

  if (SD.exists("example.txt")) {
    Serial.println("example.txt exists.");
  }
  else {
    Serial.println("example.txt doesn't exist.");
  }
}

void loop()
{
  // nothing happens after setup finishes.
}
```

## Liste de fichiers

```
/*
 Listfiles

 This example shows how print out the files in a
 directory on a SD card

 The circuit:
```

```
 * SD card attached to SPI bus as follows:
 ** MOSI - pin 11
 ** MISO - pin 12
 ** CLK - pin 13
 ** CS - pin 4

 created   Nov 2010
 by David A. Mellis
 modified 9 Apr 2012
 by Tom Igoe
 modified 2 Feb 2014
 by Scott Fitzgerald

 This example code is in the public domain.
*/

#include <SPI.h>
#include <SD.h>

File root;

void setup()
{
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }

  Serial.print("Initializing SD card...");

  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  root = SD.open("/");

  printDirectory(root, 0);

  Serial.println("done!");
}

void loop()
{
  // nothing happens after setup finishes.
}

void printDirectory(File dir, int numTabs) {
   while(true) {

     File entry =  dir.openNextFile();
     if (! entry) {
       // no more files
       break;
     }
     for (uint8_t i=0; i<numTabs; i++) {
       Serial.print('\t');
     }
     Serial.print(entry.name());
```

```
    if (entry.isDirectory()) {
      Serial.println("/");
      printDirectory(entry, numTabs+1);
    } else {
      // files have sizes, directories do not
      Serial.print("\t\t");
      Serial.println(entry.size(), DEC);
    }
    entry.close();
  }
}
```

## Carte SD lecture / écriture

```
/*
  SD card read/write

 This example shows how to read and write data to and from an SD card file
 The circuit:
 * SD card attached to SPI bus as follows:
 ** MOSI - pin 11
 ** MISO - pin 12
 ** CLK - pin 13
 ** CS - pin 4

 created   Nov 2010
 by David A. Mellis
 modified 9 Apr 2012
 by Tom Igoe

 This example code is in the public domain.

 */

#include <SPI.h>
#include <SD.h>

File myFile;

void setup()
{
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }


  Serial.print("Initializing SD card...");

  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.
  myFile = SD.open("test.txt", FILE_WRITE);
```

```
  // if the file opened okay, write to it:
  if (myFile) {
    Serial.print("Writing to test.txt...");
    myFile.println("testing 1, 2, 3.");
    // close the file:
    myFile.close();
    Serial.println("done.");
  } else {
    // if the file didn't open, print an error:
    Serial.println("error opening test.txt");
  }

  // re-open the file for reading:
  myFile = SD.open("test.txt");
  if (myFile) {
    Serial.println("test.txt:");

    // read from the file until there's nothing else in it:
    while (myFile.available()) {
      Serial.write(myFile.read());
    }
    // close the file:
    myFile.close();
  } else {
    // if the file didn't open, print an error:
    Serial.println("error opening test.txt");
  }
}

void loop()
{
  // nothing happens after setup
}
```

Lire Stockage de données en ligne: https://riptutorial.com/fr/arduino/topic/6584/stockage-de-donnees

# Chapitre 25: Utiliser Arduino avec Atmel Studio 7

## Remarques

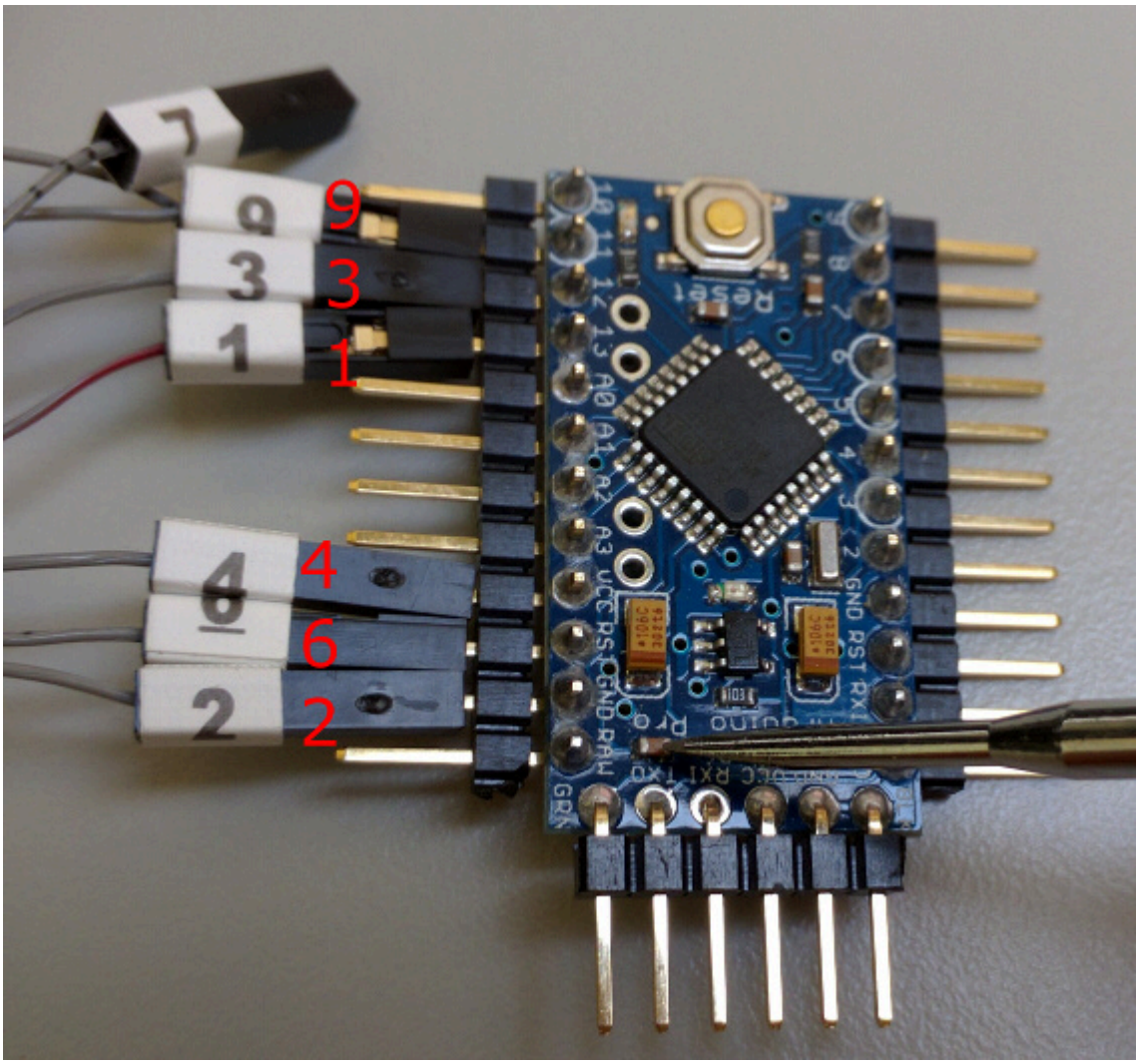
# Installer

- Téléchargez et installez Atmel Studio 7 à partir d' ici .
- Achetez un débogueur. Vous pouvez vous débrouiller avec un programmeur ISP, mais si vous voulez des fonctionnalités de débogage, ce qui est l'un des grands avantages de l'utilisation d'Atmel Studio, vous voudrez un débogueur. Je recommande le ICE d'Atmel , car il fournit des capacités de débogage pour les arduinos basés sur AVR (comme Uno, Pro mini, etc.) et les Arduinos basés sur ARM, tels que Zero et Due. Si vous avez un budget, vous pouvez l' obtenir sans le boîtier en plastique et faites attention à ne pas le choquer .

# Les liaisons

- Pour l'Uno, utilisez le câble ICSP à 6 broches . Branchez un côté dans l'Uno comme indiqué. Branchez l'autre côté dans le port AVR du débogueur.

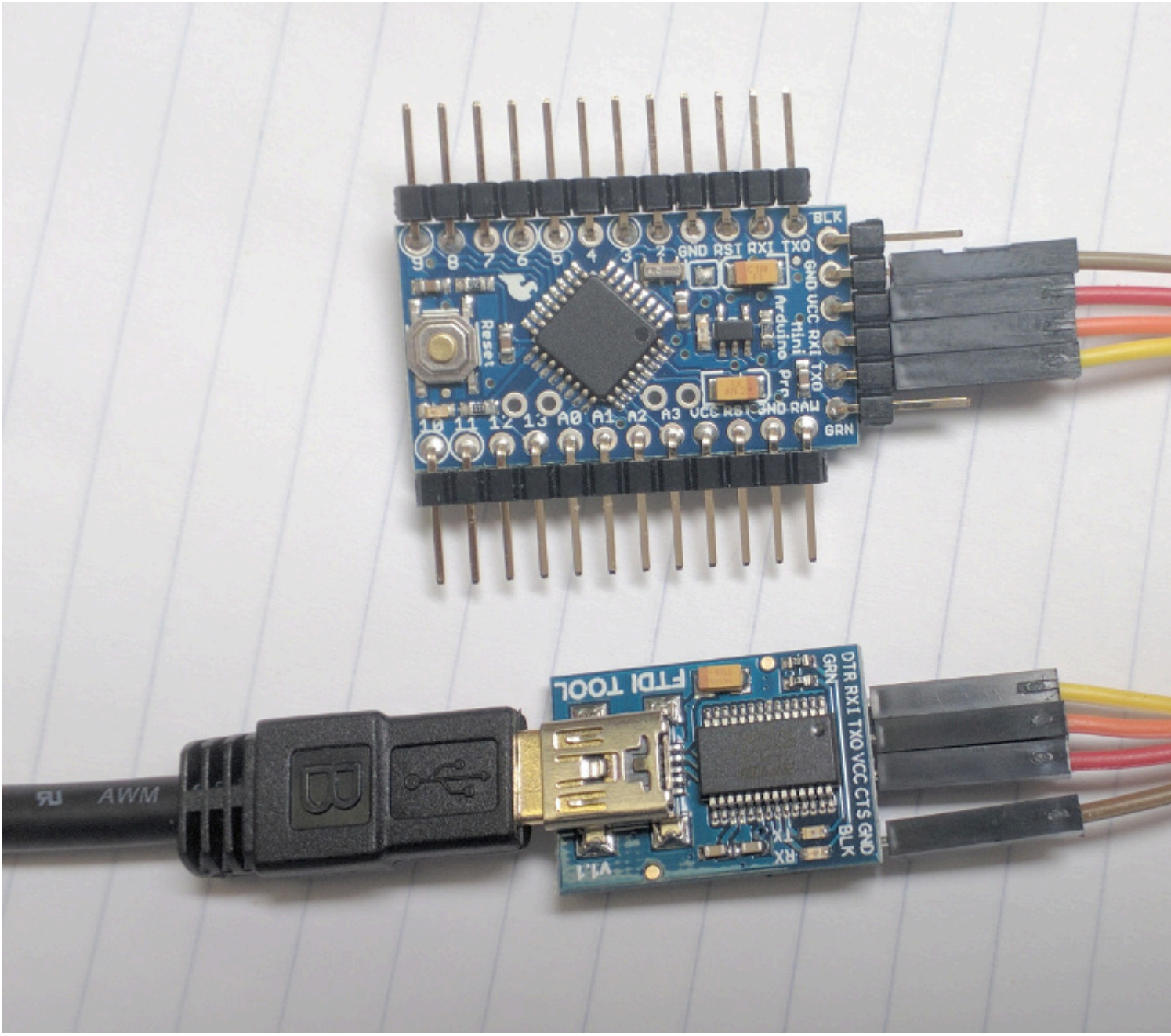

Pour l'Arduino Pro Mini, utilisez le câble mini-squid comme indiqué, en connectant à nouveau l'autre côté du port AVR du débogueur.




## Considérations de débogage

Pour le débogage avec Uno, vous devez couper le tracé d'activation de Reset (vous pouvez toujours le souder pour l'utiliser avec l'IDE Arduino):


En utilisant le Pro Mini, si vous avez l'intention de connecter le port série à votre ordinateur en utilisant une carte FTDI, ne connectez pas la ligne DTR, car cela pourrait interférer avec l'interface de débogage de fil série (SWD) d'Atmel. Je connecte simplement l'alimentation, la terre, Tx et Rx comme indiqué ci-dessous. Rx et Tx sur Arduino vont à Tx et Rx, respectivement sur la carte FTDI. Certaines cartes FTDI sont étiquetées différemment, donc si le port série ne fonctionne pas, échangez Rx et Tx.




Vous devrez fournir l'alimentation séparément à l'Arduino car le débogueur ne l'alimentera pas. Cela peut être fait sur le Pro Mini via la carte FTDI comme indiqué ci-dessus, ou avec un câble USB ou un adaptateur secteur sur le Uno.

# Configuration du logiciel

Branchez Atmel ICE sur votre ordinateur, démarrez Atmel Studio et vous pouvez désormais importer un projet Arduino existant.

Dans Atmel Studio, allez dans Fichier -> Nouveau -> Projet et sélectionnez "Créer un projet à partir d'une esquisse Arduino". Remplissez les options, y compris les menus déroulants du tableau et des périphériques.

Allez dans Project -> yourProjectName Properties, cliquez sur Tool, sélectionnez Atmel ICE sous

debugger / programmer et debugWire sous interface. Aller au débogage -> Démarrer le débogage et casser. Vous devriez voir un avertissement et vous demander si vous souhaitez définir le fusible DWEN. Choisissez OK, débranchez l'Arduino du secteur et rebranchez-le. Vous pouvez arrêter le débogage en cliquant sur le bouton carré rouge et commencer par cliquer sur le bouton triangle vert. Pour ramener l'Arduino à un état dans lequel il peut être utilisé dans l'EDI Arduino, pendant que vous êtes en train de déboguer, choisissez Debug -> disable debugWIRE and close.

Notez que toutes les fonctions que vous ajoutez doivent également inclure un prototype de fonction (la boucle et la configuration n'en ont pas besoin). Vous pouvez voir ceux que Atmel Studio a ajoutés en haut de l'esquisse s'il y avait des fonctions lorsque vous avez importé votre projet dans Atmel Studio (voir l'exemple de code par exemple).

La prise en charge de C ++ 11 est activée par défaut dans Arduino 1.6.6 et les versions ultérieures. Cela fournit plus de fonctionnalités de langage C ++ et son activation peut augmenter la compatibilité avec le système Arduinio. Pour activer C ++ 11 dans Atmel Studio 7, faites un clic droit sur votre fichier de projet, sélectionnez les propriétés, cliquez sur ToolChain sur la gauche, cliquez sur Miscellaneous sous AVR / GNU C ++ Compiler et mettez `-std=c++11` dans les autres flags champ.

## Pour inclure des bibliothèques dans votre esquisse

Copiez le fichier de bibliothèque .cpp dans `C:\Users\YourUserName\Documents\Atmel Studio\7.0\YourSolutionName\YourProjectName\ArduinoCore\src\core` core core, choisissez add -> item existant et choisissez le fichier que vous avez ajouté. Faites de même avec le fichier de bibliothèque .h et le dossier YourProjectName / Dependancies.

## Ajouter la fenêtre du terminal

Vous pouvez toujours ouvrir l'IDE Android et utiliser cette fenêtre série (sélectionnez simplement le port série approprié). Toutefois, pour ajouter une fenêtre série intégrée à Atmel Studio, accédez à Outils -> Extensions et mises à jour, cliquez sur Téléchargements disponibles et recherchez Terminal Window ou Terminal pour Atmel Studio et installez-le. Une fois installé, accédez à Affichage -> Terminal Window.

## Avantages

La programmation d'Arduino avec un IDE modéré comme Atmel Studio 7 vous offre de nombreux avantages par rapport à l'IDE Arduino, notamment le débogage, l'auto-complétion, la définition et la déclaration, la navigation avant / arrière, les signets et les options de refactorisation.

Vous pouvez configurer les liaisons de clé en allant dans Outils -> Options -> Environnement -> Clavier. Certains accélèrent le développement:

- Edit.CommentSelection, Edit.UncommentSelection
- View.NavigateForward, View.NavigateBackward
- Edit.MoveSelectedLinesUp, Edit.MoveSelectedLinesDown
- Edit.GoToDefinition

# Examples

## Exemple de croquis importé Atmel Studio 7

Ceci est un exemple de ce à quoi ressemble une simple esquisse Arduino après avoir été importée dans Atmel Studio. Atmel Studio a ajouté les sections générées automatiquement en haut. Le reste est identique au code Arduino d'origine. Si vous développez le projet ArduinoCore qui a été créé et regardez dans le dossier src -> core, vous trouverez `main.cpp` , le point d'entrée du programme. Vous pouvez y voir l'appel à la fonction de configuration Arduino et une boucle sans fin pour appeler la fonction de boucle Arduino encore et encore.

```
/* Begining of Auto generated code by Atmel studio */
#include <Arduino.h>
/* End of auto generated code by Atmel studio */


// Beginning of Auto generated function prototypes by Atmel Studio
void printA();
// End of Auto generated function prototypes by Atmel Studio

void setup() {
  Serial.begin(9600);
}

void loop() {
    printA();
}

void printA() {
    Serial.println("A");
}
```

Lire Utiliser Arduino avec Atmel Studio 7 en ligne:
https://riptutorial.com/fr/arduino/topic/2567/utiliser-arduino-avec-atmel-studio-7

# Chapitre 26: Variables et types de données

## Examples

### Créer une variable

Pour créer une variable:

```
variableType variableName;
```

Par exemple:

```
int a;
```

Pour créer une variable et l'initialiser:

```
variableType variableName = initialValue;
```

Par exemple:

```
int a = 2;
```

### Attribuer une valeur à une variable

Si vous avez une variable déclarée auparavant, vous pouvez lui attribuer une valeur:

Par exemple:

```
int a;  // declared previously
a = 2;
```

Ou changez la valeur:

```
int a = 3;  // initalized previously
a = 2;
```

### Types de variables

- `char` : valeur de 1 octet signée
- `byte` : entier non signé de 8 bits
- `int` : signé 16 bits (sur des cartes ATMEGA) ou 32 bits (sur Arduino Due)
- `unsigned int` : entier non signé 16 bits (sur les cartes ATMEGA) ou 32 bits (sur Arduino Due)
- `long` : entier de 32 bits signé
- `unsigned long` : entier non signé de 32 bits
- `float` : nombre à virgule flottante de 4 octets

- `double` : 4 octets (sur les cartes ATMEGA) ou 8 octets (sur Arduino Due)

Exemples:

```
char a = 'A';
char a = 65;

byte b = B10010;

int c = 2;

unsigned int d = 3;

long e = 186000L;

unsigned long f = millis(); // as an example

float g = 1.117;

double h = 1.117;
```

Lire Variables et types de données en ligne: https://riptutorial.com/fr/arduino/topic/2565/variables-et-types-de-donnees

# Crédits

| S. No | Chapitres | Contributeurs |
|---|---|---|
| 1 | Démarrer avec Arduino | Abhishek Jain, Christoph, Community, Danny_ds, Doruk, geek1011, gmuraleekrishna, H. Pauwelyn, jleung513, Martin Carney, Mizole Ni, Shef, uruloke, Wolfgang |
| 2 | Arduino IDE | geek1011, Jeremy, jleung513, sohnryang, uruloke |
| 3 | Bibliothèque de cristaux liquides | Morgoth |
| 4 | Bibliothèques | Oscar Lundberg |
| 5 | Boucles | datafiddler, Martin Carney, MikeCAT |
| 6 | Broches de matériel | Jeremy, Martin Carney |
| 7 | Comment Python s'intègre à Arduino Uno | Danny_ds, Peter Mortensen, Stark Nguyen |
| 8 | Comment stocker des variables dans EEPROM et les utiliser pour le stockage permanent | AZ Vcience, Chris Combs, Danny_ds, Jeremy, Peter Mortensen , RamenChef |
| 9 | Communication Bluetooth | Girish, Martin Carney |
| 10 | Communication I2C | Asaf |
| 11 | Communication MIDI | Rich Maes |
| 12 | Communication série | blainedwards8, Danny_ds, geek1011, Leah, Martin Carney, MikeS159, Morgoth, Nufail Achath, Peter Mortensen, uruloke |
| 13 | Communication SPI | Christoph |
| 14 | Entrées analogiques | Jake Lites, MikeS159, Ouss4, uruloke |
| 15 | Entrées Numériques | Martin Carney, uruloke |
| 16 | Gestion du temps | Christoph, Rei |

| | | |
|---|---|---|
| 17 | Les fonctions | datafiddler, Leah, MikeCAT |
| 18 | Les interruptions | DavidJ, Martin Carney |
| 19 | Nombres aléatoires | Danny_ds, Javier Rizzo Aguirre, MikeCAT |
| 20 | PWM - Modulation de la largeur d'impulsion | Danny_ds, Johnny Mopp, JorgeGT, Martin Carney |
| 21 | Servo | geek1011, mactro, Morgoth |
| 22 | Sortie audio | Jake Lites, MikeCAT |
| 23 | Sortie numérique | uruloke |
| 24 | Stockage de données | Danny_ds, robert |
| 25 | Utiliser Arduino avec Atmel Studio 7 | Danny_ds, Nate |
| 26 | Variables et types de données | Leah, MikeCAT |