Introduction à Arduino

VOUS AVEZ DIT ARDUINO ?

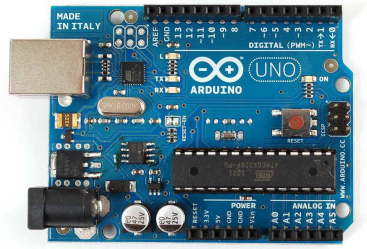
- Pont tendu entre le monde réel et le monde numérique, Arduino permet d'étendre les capacités de relations humain/machine ou environnement/machine.
- Arduino est un projet en source ouverte (open source) : la communauté importante d'utilisateurs et de concepteurs permet à chacun de trouver les réponses à ses questions.

VOUS AVEZ DIT ARDUINO ?

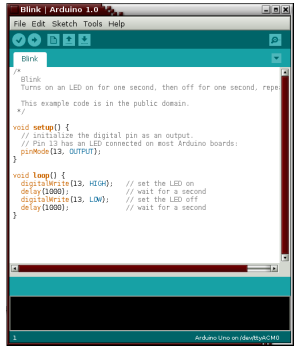
- Arduino est une plate-forme de prototypage d'objets interactifs à usage créatif constituée d'une carte électronique et d'un environnement de programmation.
- Sans tout connaître ni tout comprendre de l'électronique, cet environnement matériel et logiciel permet à l'utilisateur de formuler ses projets par l'expérimentation directe avec l'aide de nombreuses ressources disponibles en ligne.

Arduino en résumé

Une carte électronique

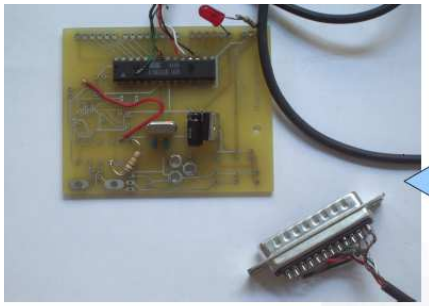


Un environnement de programmation

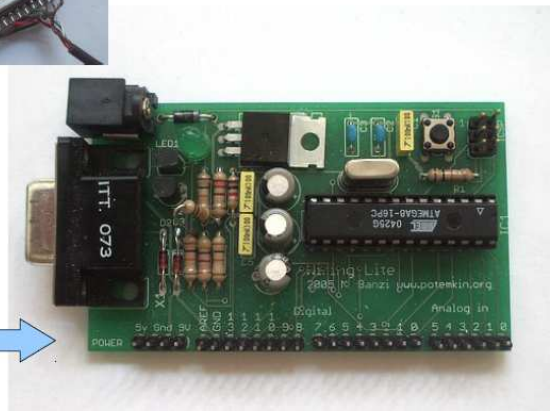


Une communauté qui échange
http://arduino.cc/

Historique



Arduino Prototype 0



Arduino Prototype 1

Leur objectif : Processing pour le Hardware !

- Qu'est ce que Processing ?
  - un langage de programmation et un environnement de développement créé par Benjamin Fry et Casey Reas, deux artistes américains.
  - particulièrement adapté à la création plastique et graphique interactive
  - Le logiciel fonctionne sur Macintosh, sous Windows et sous Linux, car il est basé sur la plate-forme Java — il permet d'ailleurs de programmer directement en langage Java.

- Pourquoi ?
  - Matériel robotique excessivement cher

Les créateurs : des artistes au sens premier du terme

Interaction Design Institute Ivrea : Team Arduino

- David Mellis
- Tom Igoe
- Gianluca Martino
- David Cuartielles
- Massimo Banzi

Arduino : une philosophie

- Le matériel est « open source » :
  - On peut le copier, le fabriquer et le modifier librement.

open hardware

open source

copyleft

- Le logiciel est libre :
  - On peut l'utiliser et le modifier librement.

- Sur l'Internet, on trouve :
  - Une communauté d'utilisateurs.
  - Des guides d'utilisation.
  - Des exemples.
  - Des forums d'entraide.

Avantages

- Pas cher !
- Environnement de programmation clair et simple.
- Multiplate-forme : tourne sous Windows, Macintosh et Linux.
- Nombreuses bibliothèques disponibles avec diverses fonctions implémentées.
- Logiciel et matériel open source et extensible.
- Nombreux conseils, tutoriaux et exemples en ligne (forums, site perso, etc.)
- Existence de « shield » (boucliers en français)

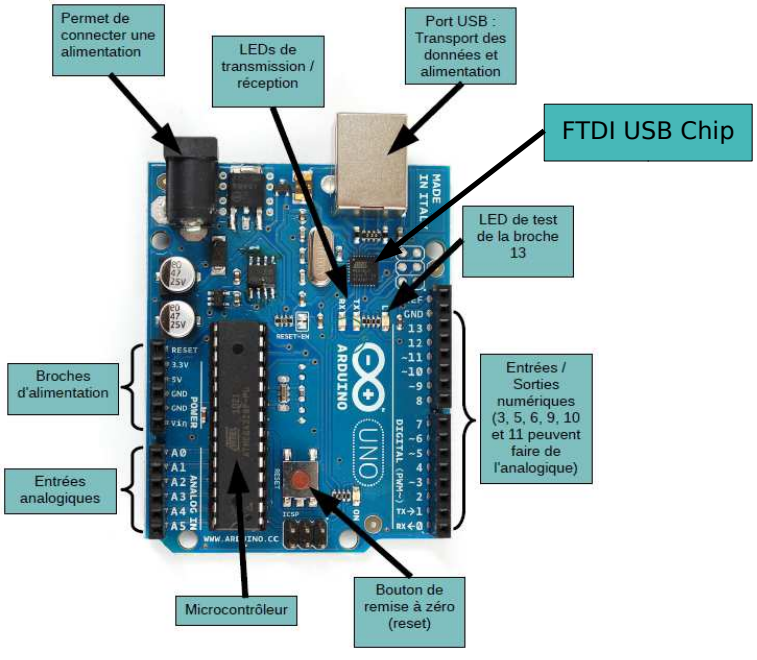
C'est quoi « pas cher » ?

- Prix d'une carte Arduino Uno = 25 euros
- Logiciel = 0 euros
- Support et assistance = 0 euros (forums)

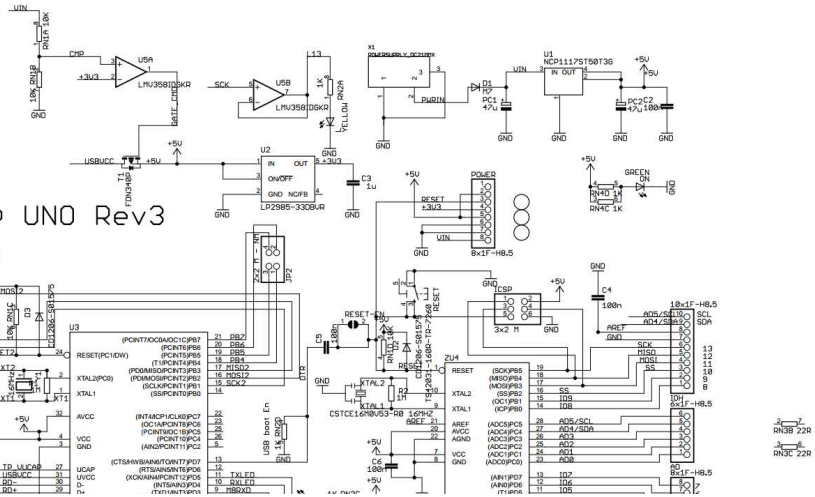
Domaine d'utilisation

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

La carte électronique Arduino

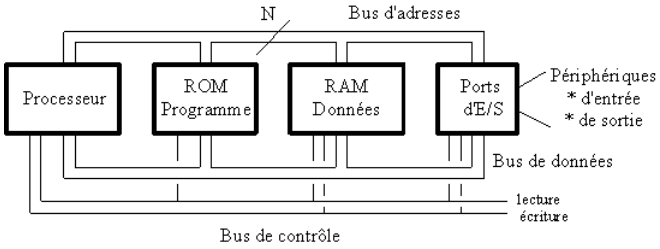


La schématique électronique



Qu'est ce qu'un microcontrôleur ?

- Un microcontrôleur intègre sur un unique die (circuit intégré) :
- un processeur (CPU), avec une largeur du chemin de données allant de 4 bits pour les modèles les plus basiques à 32 ou 64 bits pour les modèles les plus évolués ;
- de la mémoire vive (RAM) pour stocker les données et variables ;
- de la mémoire pour stocker le programme : ROM (mémoire morte) et/ou EPROM, EEPROM, Flash ;
- souvent un oscillateur pour le cadencement. Il peut être réalisé avec un quartz, un circuit RC ou encore une PLL[1] ;



1 : Phase-Locked Loop ou boucle à phase asservie, ou encore boucle à verrouillage de phase

Qu'est ce qu'un microcontrôleur ?

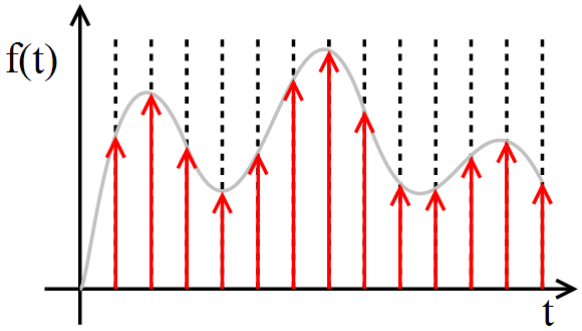
- µcontrôleur : circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires (mémoire morte et/ou programmable pour le programme, mémoire vive pour les données), unités périphériques et interfaces d'entrées-sorties
- Ils sont fréquemment utilisés dans les systèmes embarqués, comme les contrôleurs des moteurs automobiles, les télécommandes, les appareils de bureau, l'électroménager, les jouets, la téléphonie mobile, etc.

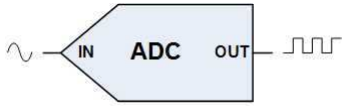
Qu'est ce qu'un microcontrôleur ?

- des périphériques, capables d'effectuer des tâches spécifiques. On peut mentionner entre autres :
  - les convertisseurs analogiques-numériques (CAN) (donnent un nombre binaire à partir d'une tension électrique),
  - les convertisseurs numériques-analogiques (CNA) (effectuent l'opération inverse),
  - les générateurs de signaux à modulation de largeur d'impulsion (MLI, ou en anglais, PWM pour Pulse Width Modulation),
  - les timers/compteurs (compteurs d'impulsions d'horloge interne ou d'événements externes),
  - les chiens de garde (watchdog),
  - les comparateurs (comparent deux tensions électriques),
  - les contrôleurs de bus de communication (UART, I²C, SSP, CAN, FlexRay, USB, Ethernet, etc.).

CAN = ADC (Analog-to-Digital Converter)
CNA = DAC (Digital-to-Analog Converter)
liaison série = UART (Universal Asynchronous Receiver Transmitter)

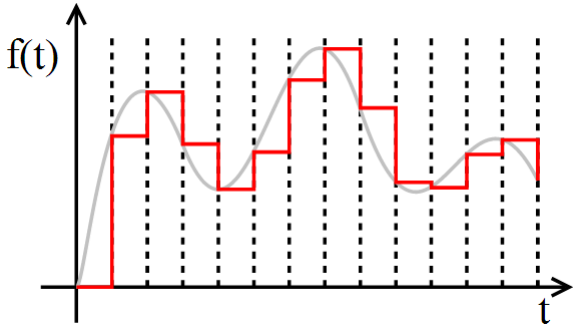
Analogique vers numérique





Résolution : le nombre de niveaux de sortie que l'ADC peut reproduire

Fréquence d'échantillonnage : le nombre de mesures par unité de temps



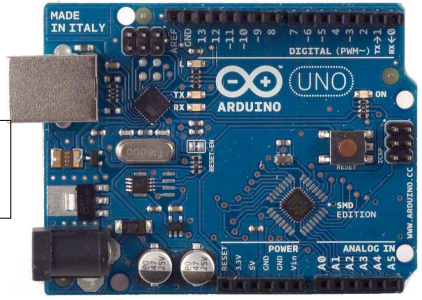
Par exemple avec une résolution de 8 bits sur un signal variant entre 0V et 5V, le nombre de niveaux est de 2^8 et donc la résolution en volt est de (5-0)/(2^8) soit environ 19,53125 mV

En pratique, on considère que les bits de poids faibles ne sont pas assez précis. Si on omet 2 bits, on arrive à 78,125mV

Notre Arduino : le Uno

- Micro contrôleur : ATmega328
- Tension d'alimentation interne = 5V
- tension d'alimentation (recommandée)= 7 à 12V, limites =6 à 20 V
- Entrées/sorties numériques : 14 dont 6 sorties PWM
- Entrées analogiques = 6 (avec une résolution de 10 bits => 1024 valeurs différentes)
- Courant max par broches E/S = 40 mA
- Courant max sur sortie 3,3V = 50mA
- Mémoire Flash 32 KB dont 0.5 KB utilisée par le bootloader*
- Mémoire SRAM 2 KB
- mémoire EEPROM 1 KB
- Fréquence horloge = 16 MHz
- Dimensions = 68.6mm x 53.3mm

Arduino Uno SMD (ou CMS en français pour Composant Monté en Surface)



La carte s'interface au PC par l'intermédiaire de sa prise USB.

La carte s'alimente par le jack d'alimentation (utilisation autonome) mais peut être alimentée par l'USB (en phase de développement par exemple).

*Bootloader : un petit programme chargé sur le microcontrôleur. Il permet de charger le code sans programmateur. Il est activé quelques secondes lorsque la carte est « *resetée* ». Ensuite, il démarre le sketch (programme) qui a été chargé sur le microcontrôleur.

Numérisation

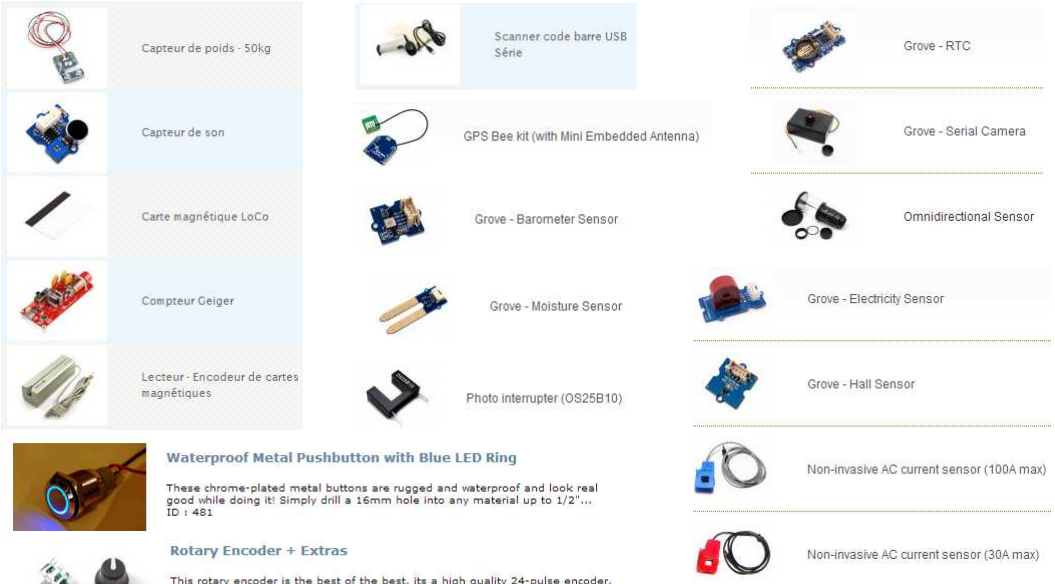
éch = échantillon ou sample en anglais

- Prenons un exemple pratique:
  - une fréquence d'échantillonnage de 10Kéch./s
  - une résolution de 10 bits (par échantillon)
  - une mémoire RAM de 2Ko
    - 10Kéch./s, soit 100Kbits/s soit 12,5Ko/s
    - Ce qui signifie qu'on ne peut récupérer que 2/12,5=160ms
- Les solutions ?
  - Réduire la fréquence d'échantillonnage si le phénomène à observer ne requière pas une telle précision temporelle
  - Réduire la résolution si une approximation plus grande est admise
  - Augmenter la mémoire :
    - si c'est une mémoire rapide, pas de problème
    - si c'est un support externe, plus lent, on va rater des événements (des mesures) le temps d'écrire.

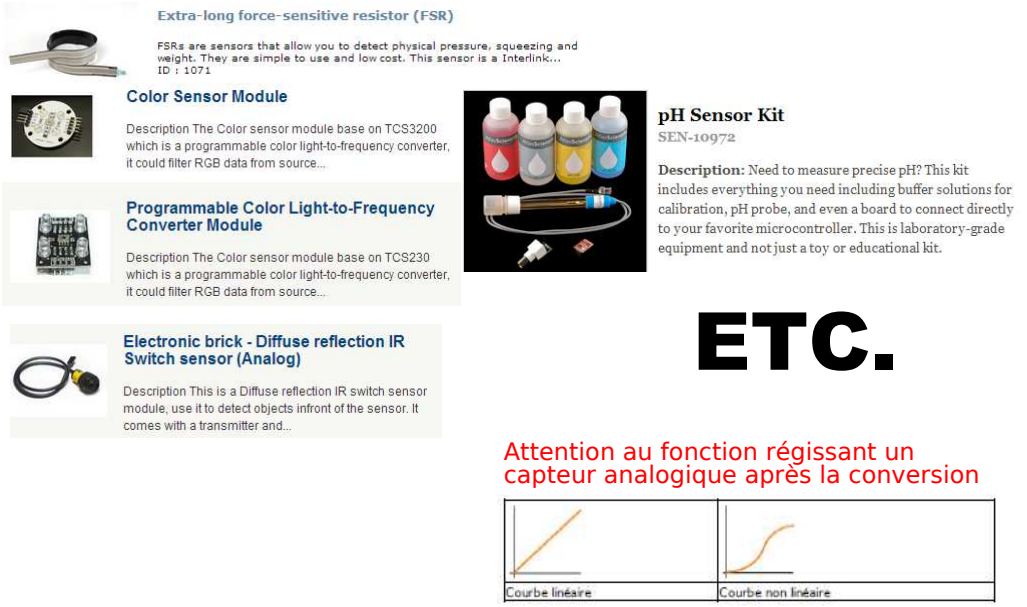
Heureusement, pour nous, la température à observer variera lentement.

Des capteurs

Encore des capteurs



Toujours des capteurs



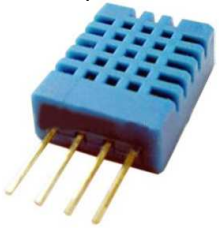
ETC.

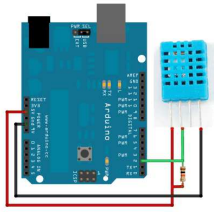
Attention au fonction régissant un capteur analogique après la conversion

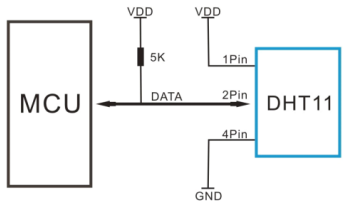
Capteurs analogiques et numériques

- Parmi les capteurs que nous venons de voir, il existe :
  - des capteurs analogiques pour lesquels le signal devra être numérisé par le CAN du microcontrôleur. Il nous appartiendra de faire appliquer la loi régissant la mesure.
  - des capteurs numériques qui ont leur propre CAN embarqué. Il gère eux même la loi régissant la mesure. La communication avec ces capteurs se fait souvent selon un protocole particulier (I2C, 1-wire, etc.).

Par exemple le DHT11



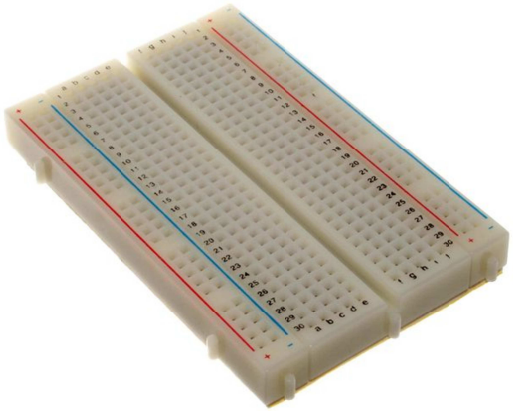


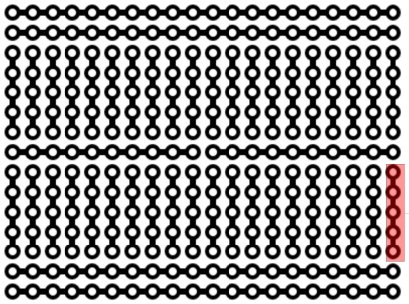


Quelques actionneurs

Breadboard (Planche à pain ... pour le prototypage)

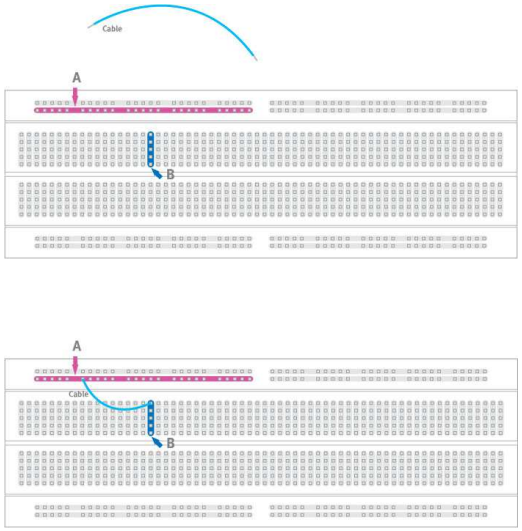
Élément essentiel pour le prototypage et essai en tout genre

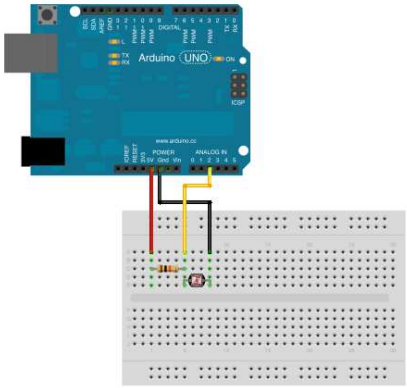




Les points reliés ensembles sont au même potentiel

Breadboard : exemple d'utilisation

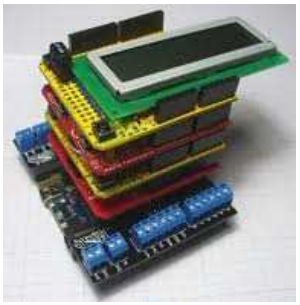




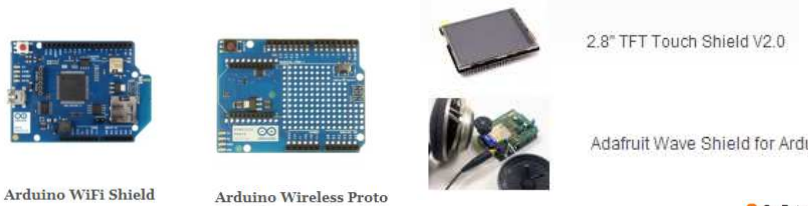
1 photorésistance

Différents shields

- Existence de « shields » (boucliers en français) : ce sont des cartes supplémentaires qui se connectent sur le module Arduino pour augmenter les possibilités comme par exemple : afficheur graphique couleur, interface ethernet, GPS, etc...



Des shields



Encore des shields

SD card shield

Solar Charger Shield V2

ETC.

RGB LED Shield V1.4 Kit

NFC Shield V2.0

LoL Shield RED - A charlieplexed LED matrix kit for the Arduino

The LoL Shield is a charlieplexed LED matrix for the Arduino. The LEDs are individually addressable, so you can use it to display anything in a 9 x...
ID : 274

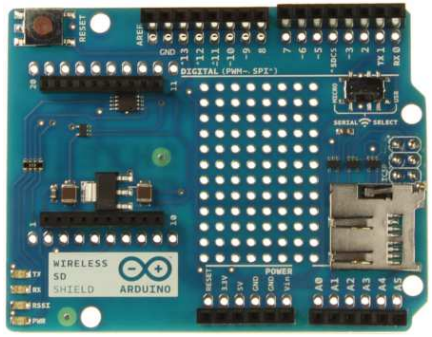
20 IN STOCK

LCD Shield Kit w/ 16x2 Character Display - Only 2 pins used!

This new Adafruit shield makes it easy to use a 16x2 Character LCD. We really like the Blue & White 16x2 LCDs we stock in the shop. Unfortunately,...
ID : 772

Le shield Wireless SD

http://arduino.cc/en/Main/ArduinoWirelessShield



Carte Micro SD 1GB + adaptateur SD

Communications sur le shield

Wifi Bee

Bluetooth Bee

XBee PCB Antenna - S1 (802.15.4)

XBee Wi-Fi PCB Antenna - S6

En général on fonctionne sur une communication série (RX/TX).
On utilisera la bibliothèque Serial ou SoftwareSerial sur Arduino.

C'est très simple !

Et ici c'est modulaire !

GPS Bee kit (with Mini Embedded Antenna)

Différents autres modules

JY-MCU Arduino Bluetooth Wireless Serial Port Module

The **Bluetooth** wireless serial port module, Drop-in replacement for wired serial connections, transparent usage. You can use it. It is simply for a serial port replacemen

free shipping

(77 reviews)

Wii WiiChuck Nunchuck Adapter for **Arduino** - Blue

ModelN/AQuantity1ColorBlueMaterialPCBFeaturesUse this shield to achieve communication between nunchuck and ArduinoApplicationDIY projectPacking List1 x **Wii** wiiChuck adap

free shipping

(4 reviews)

Arduino Compatible NRF24L01 + PA + LNA V3.1 **Wireless** Module

ModelN/AQuantity1ColorBlack + GoldenMaterialFR4FeaturesUnder 250K transmission rate, effective range: 1000m (open land); 1M transmission rate, 750m; 2M transmission rate, 520mApplicationArduino DIY pr

free shipping

NRF905 Wireless Communication Transmission Module for Arduino

ModelI061808Quantity1ColorGreenMaterialCCLFeaturesOperating frequency: 433/868/915MHz; Number of channel: 170; Modulation Mode: FSK/GMSK; Max output power: +10dBm; Sensitivity: -100dBm; Max operating

free shipping

DIY 433MHz Wireless Receiving Module for Arduino - Green

+ PCBFeaturesOperating voltage: DC 5V; Quiescent current: 4mA; Modulation mode: OOK; Receiving sensitivity: -105dBm; Operating frequency: 433.90MHzApplicationVarious types of electronic products or DI

free shipping

NRF2401B 2.4GHz Wireless RF Transceiver Module

PCB - With RF2401B **wireless** communication module - 2.4GHz ISM frequency range - Up to 1Mbps working speed - 125 channels - High anti-jamming GFSK modulation - 8 / 16-bit

free shipping

» **Arduino** & SCM Supplies

315Mhz RF link kit

315Mhz RF link kits - with encoder and decoder

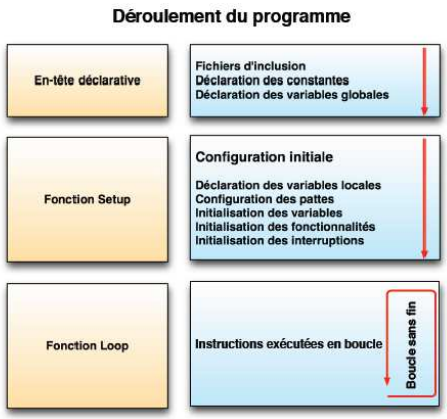
PLL FM(88~108MHz) Stereo Audio Receiver Module

PLL FM(88~108MHz) Stereo Audio Transmitter Module

Programmons notre Arduino

- Le langage Arduino est basé sur le C/C++.
  - Le langage de programmation d'Arduino est en effet une implémentation de Wiring (une plate-forme open source similaire de *physical computing* qui proposait elle-même une bibliothèque appelée **Wiring** qui simplifie les opérations d'entrée/sortie).
- Un programme Arduino est aussi appelé un sketch.

Structure d'un programme



- Prise en compte des instructions de la partie déclarative
- Exécution de la partie configuration (`fonction setup()`),
- Exécution de la boucle sans fin (`fonction loop()`): le code compris dans la boucle sans fin est exécuté indéfiniment.

Finalement similaire au code suivant sur PC :

```
int main()
{
    setup();
    while(true)
        loop();
}
```

Le code minimal

- Avec Arduino, nous devons utiliser un code minimal lorsque l'on crée un programme. Ce code permet de diviser le programme que nous allons créer en deux grosses parties.

```
void setup() //fonction d'initialisation de la carte
{
            //contenu de l'initialisation
}


void loop() //fonction principale qui se répète (s'exécute) à l'infini
{
           //contenu du programme
}
```

La syntaxe du langage

- Voir aussi la section reference

  http://www.arduino.cc/reference

Program structure

void setup() { . . . } void loop() { . . . }

Statements

; {}

// /* */

#define #include

Control structures

if, if...else, for, switch case, while, do... while

break, continue, return, goto

Control structures

type func(type param, ...)

La syntaxe du langage

Data Types

void, boolean, char, unsigned char, byte,
int, unsigned int, word, long, unsigned long
float, double, string (char[])

Contructor

[]

Variables

local, global, static local, volatile, const, sizeof()

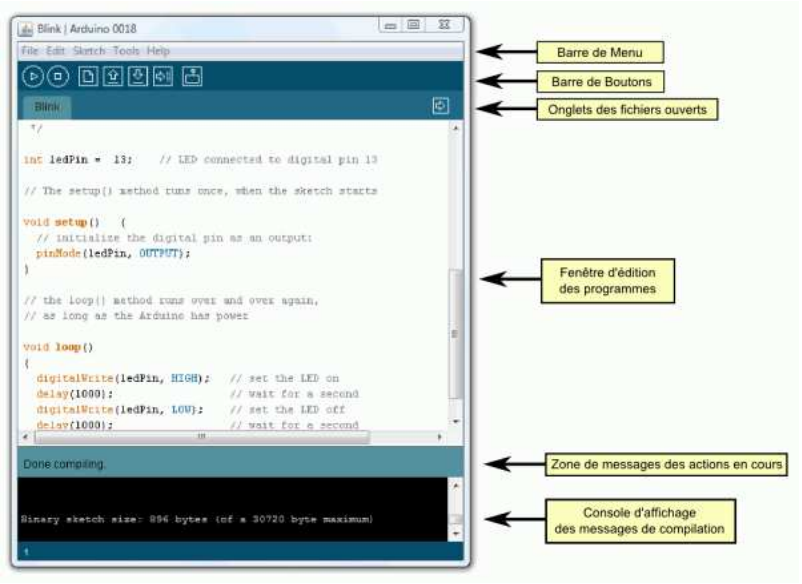
Operators

Arithmetic = + - * / %
Comparison == != < > <= >=
Boolean && || !
Bitwise & | ^ ~ « »
Pointer Access * &
Compound == -- += -= *= /=&= |=

L'environnement de développement

- Vérifier (Verify) : vérifier les erreurs dans le code
- Charge (Upload) : compiler le code et charge le programme sur la carte Arduino
- Nouveau (New) : créer un nouveau sketch
- Ouvrir (Open) : ouvrir un des sketchs déjà présent
- Sauvegarder (Save) : sauvegarder le sketch
- Serial Monitor : permet d'accéder au port série (en RX/TX)

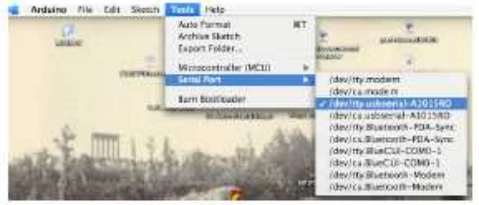
L'environnement de développement



Attention la barre d'outils n'est pas la même dans notre version !

Configuration de l'environnement de développement

- Pour un Arduino Uno dans Microcontroller (ou « type de carte » suivant les versions d'IDE)
- Désigner le bon port Série



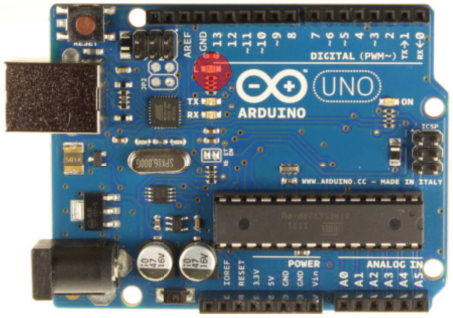
Sur MacOSX
(les chiffres après tty.usbserial seront différents)

Si vous changez de carte arduino, il faudra ré-indiquer le bon port.

Sur Windows

Un premier exemple : la led clignotante sur la carte

```
/*
Objectif: faire clignoter la LED montée sur la carte (et reliée à la patte 13)
*/

void setup()
{
    // Initialise la patte 13 comme sortie
    pinMode(13, OUTPUT);
}

void loop()
{
    digitalWrite(13, HIGH); // allume la LED
    delay(500); // attend 500ms
    digitalWrite(13, LOW); // éteint la LED
    delay(500); // attend 500ms
}
```

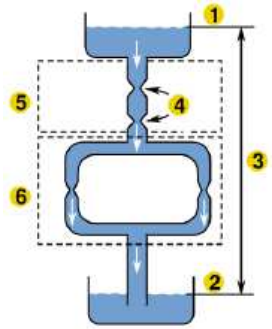


Quelques explications

- La ligne pinMode(13, OUTPUT); initialise la patte 13 du microcontrôleur comme sortie, c'est-à-dire que des données seront envoyées depuis le microcontrôleur vers cette patte.
- Avec l'instruction digitalWrite(13, HIGH);, le microcontrôleur connecte la patte D13 au +5V ce qui a pour effet d'allumer la LED.
- L'instruction delay(500); indique au microcontrôleur de ne rien faire pendant 500 millisecondes, soit ½ seconde.
- Avec l'instruction digitalWrite(13, LOW);, le microcontrôleur connecte la patte D13 à la masse (Gnd) ce qui a pour effet d'éteindre la LED.
- L'instruction delay(500); indique au microcontrôleur à nouveau de ne rien faire pendant 500ms soit ½ seconde.

Notions électriques fondamentales

- La tension est la différence de potentiel (Volts, V)
- Le courant (Ampères, A)
  - équivalent du débit
- La résistance (Ohms, Ω)
  - Équivalent du rétrécissement
  - U=R*I
- AC/DC
  - Direct Current (Courant Continu) (en électronique)
  - Alternative Current (Courant Alternatif)
    - courant qui change de direction continuellement
    - peut être périodique, c-à-d que sa fréquence est constante.
    - Souvent sinusoïdal, il est caractérisé par sa fréquence notée f et exprimée en Hertz, qui correspond au nombre d'aller-retour par seconde



D'où

- DANGER

- Pas de danger pour nous avec la basse tension

Même une immense retenu d'eau peu profonde qui se déverserait à grand débit n'est pas un problème

Limites électroniques

- Nous allons confronter la théorie à l'épreuve de la réalité

La carte Arduino est une petite chose fragile, il convient de bien en comprendre ses limites d'utilisation et vérifier la compatibilité avec les matériels que l'on souhaite y raccorder...

Tension :

le microcontrôleur placé sur la carte est prévu pour fonctionner entre 3,3V et 5V.

Intensité :

le courant de sortie de chaque broche (D0 à D13) ne doit pas dépasser 40mA

le courant issu du port USB ne doit pas dépasser 500mA

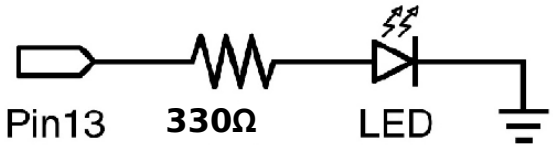
le courant soutiré à la broche « 3,3V » ne doit pas dépasser 50mA

Les différents niveaux de protections

- Protection du port USB
- Protection de la carte Arduino
  - Protection des entrées numériques
- Protection des composants

Un second exemple : clignotement d'une led externe

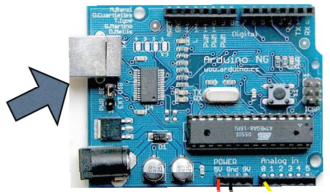
- **Attention**, on ne branche pas n'importe quoi n'importe comment.



- **Pas** de moteur/lampe, de composants nécessitant de la puissance directement connecté à un port d'entrée/sortie qui est **en mode sortie** !

Protection du port USB

- Tout d'abord, il faut savoir que le port USB de votre ordinateur délivre une quantité limitée de courant électrique. En général un port USB peut fournir au maximum 500 mA en 5 V.
- Si vous souhaitez réaliser un montage qui nécessite plus de courant, il s'agira de prévoir une alimentation externe suffisamment puissante pour répondre à votre besoin.



- Conseils :
  - Pour éviter qu'un fil ou qu'un composant branché au + vienne endommager un port USB dans l'ordinateur, isoler le métal du port USB avec un adhésif d'électricien
  - Attention également au dessous de la carte, à ne pas poser sur un support conducteur.

Protection de la carte Arduino

- Sachez que chacune des entrées/sorties de la carte ne peut pas délivrer plus de 40 mA (pour nos cartes Arduino Uno – voir les spécifications sur http://arduino.cc/en/Main/ArduinoBoardUno).
- Cette quantité de courant est relativement faible mais permet, par exemple, de contrôler une diode électroluminescente DEL (ou LED en anglais) ainsi que des actionneurs de faible puissance tel qu'un piézoélectrique ou encore un petit servomoteur.

La protection des entrées numériques sera vue plus loin.

Circuits de commande et de puissance

- Lorsqu'un système comporte des moteurs ou des actionneurs demandant un certain niveau de puissance que le micro-contrôleur ne peut pas fournir, il faut avoir recours à deux circuits distincts interconnectés, appelés « circuit de commande » et « circuit de puissance ».
- Suivant la puissance présente, les interconnexions entre les deux circuits nécessiteront des précautions particulières.
- Il est important de noter que ces deux circuits sont montés distinctement et sont isolés l'un de l'autre.
  - Toutefois, il est primordial de lier leur masse (« Gnd ») afin qu'ils partagent le même potentiel de référence.

Protection des composants

- Chaque composant possède ses propres conventions d'utilisation.
- Par exemple, il existe des composants qui possèdent un sens de branchement à respecter. On dit que ces composants sont polarisés.
  - C'est le cas des LEDs, de certains condensateurs, des diodes, etc.
- La plupart des composants ne peuvent pas fonctionner seuls, par exemple une LED a besoin d'une résistance appropriée pour ne pas s'user ou « brûler ». Cette résistance permet de limiter le courant qui traverse la LED. Le courant supprimé est alors dissipé en chaleur par la résistance (« Loi d'Ohm »).

Circuit de commande

- C'est dans ce circuit que sont rassemblés tous les éléments de contrôle comme les boutons, les interfaces et le micro-contrôleur.
  - Il est alimenté en basse tension : moins de 50V, souvent 12V, ou avec la carte Arduino 5V.
- On pourrait l'assimiler au système nerveux d'un organisme : c'est ici que se prennent les décisions mais peu d'énergie y circule.

Circuit de commande

- La manière la plus simple de relayer les commandes émergeant de ce circuit pour les transmettre au circuit de puissance est d'utiliser des transistors ou encore des relais.
- Lorsque les tensions d'alimentation des deux circuits sont plus importantes ou si l'on veut protéger la commande de retours accidentels de courant provenant de la puissance, des optocoupleurs (plutôt que des transistors) assurent une isolation galvanique : l'information est transmise sous forme de lumière. Ainsi, les deux circuits sont complètement isolés électriquement.

Courts-circuits

- Votre carte ne doit pas être posée sur un support conducteur car elle possède sur son verso des zones nues qui ne doivent pas êtres mises en contact afin de ne pas court-circuiter les composants entre eux.
- Il faut aussi ne jamais connecter directement le port noté « Gnd » (pôle négatif) avec la broche 5 V (pôle positif).

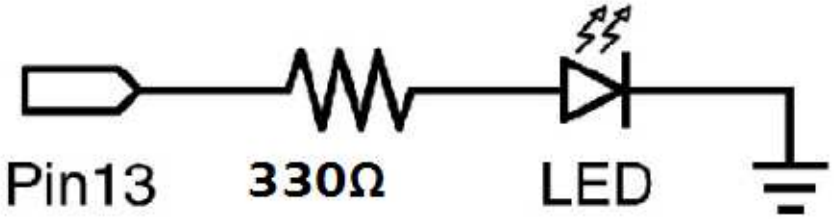
Circuit de puissance

- Ce circuit alimente les composants nécessitant beaucoup d'énergie (habituellement les moteurs et autres actionneurs). Sa tension d'alimentation dépend des composants en question.
- En fonction de cette tension et des conditions d'utilisation, les précautions à prendre sont variables :
  - dans tous les cas, une protection contre les courts-circuits est conseillée : fusible ou disjoncteur pour éviter de détruire les composants ;
  - **au dessus de 50 V, la tension peut menacer directement les humains** : protéger les pièces nues sous tension ;
  - le 230 V nécessite un interrupteur différentiel qui protège les humains des contacts accidentels (en général tout local est doté de ce dispositif). **Ne pas manipuler de 230 V sans connaissances appropriées.**

Quelques conseils (qu'on ne suivra pas tous ...)

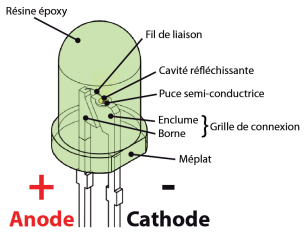
- Certains composants se ressemblent mais n’ont pas du tout la même fonction : toujours bien regarder leur signalétique.
- Ne pas manipuler de 240V ou 110 V sans connaissances appropriées.
- Préférer faire les essais et les montages avec une alimentation externe plutôt que celle de l’USB (humm), ce qui évitera de griller votre port USB de l’ordinateur (très courant)
- À NE PAS FAIRE : 10 façons de tuer son Arduino
  - http://ruggedcircuits.com/html/ancp01.html

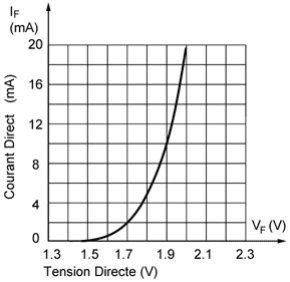
Retour sur le second exemple : clignotement d'une led externe



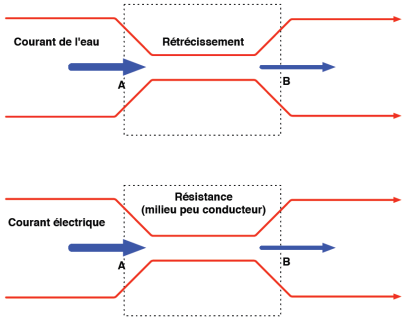
Led

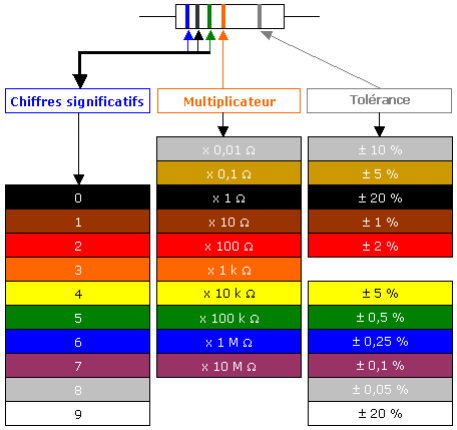
- Une diode électroluminescente (DEL ou LED) est un composant optoélectronique capable d'émettre de la lumière lorsqu'il est parcouru par un courant électrique.
- Une diode électroluminescente ne laisse passer le courant électrique que dans un seul sens (le sens passant).
- lorsqu'elle est traversée par un courant, la LED oppose une tension fixe
- L'intensité admissible dans la LED est aussi limitée (pour les LED ordinaires de 5mm, 24mA environ)



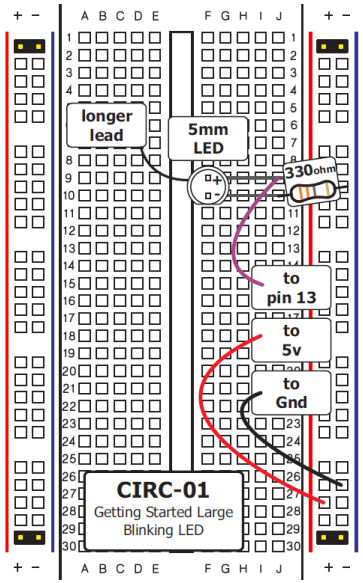


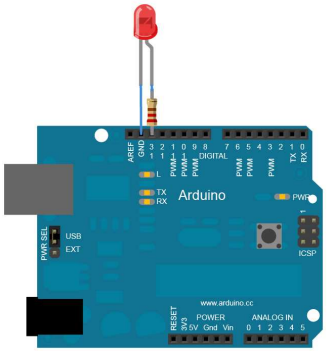
Résistance

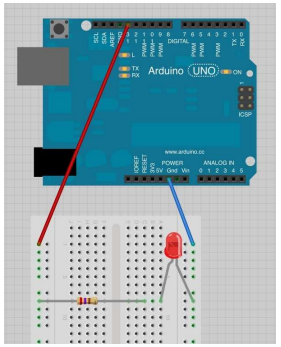




Le circuit du second exemple







Le code du second exemple

Commentaires
Toujours écrire des commentaires sur le programme: soit en multiligne, en écrivant entre des /*****/, soit sur une ligne de code en se séparant du code avec //

Définition des variables:
Pour notre montage, on va utiliser une sortie numérique de la carte, qui est par exemple la 13 ème sortie numérique. Cette variable doit être définie et nommée ici: on lui donne un nom arbitraire BrocheLED . Le mot de la syntaxe est pour désigner un nombre *entier* est int

Configuration des entrées-sorties void setup():
Les broches numériques de l'Arduino peuvent aussi bien être configurées en entrées numériques ou en sorties numériques. Ici on va configurer BrocheLED en sortie.
pinMode (nom, état) est une des quatre fonctions relatives aux entrées-sorties numériques.

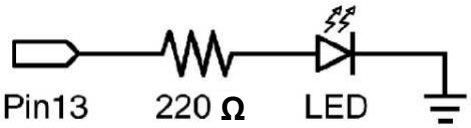
Programmation des interactions void loop():
Dans cette boucle, on définit les opérations à effectuer, dans l'ordre:
• digitalWrite (nom, état) est une autre des quatre fonctions relatives aux entrées-sorties numériques.
•delay(temps en millisecondes) est la commande d'attente entre deux autres instruction
•Chaque ligne d'instruction est terminée par un point virgule
•Ne pas oublier les accolades, qui encadrent la boucle.

(Syntaxe en marron, paramètres utilisateur en vert)

```
/* Ce programme fait clignoter une LED branchée sur la broche 13
 * et fait également clignoter la diode de test de la carte
 */


int BrocheLED = 13;     // Définition de la valeur 13 et du nom de la broche à utiliser



void setup()
{
  pinMode(BrocheLED, OUTPUT);      // configure BrocheLED comme une sortie
}


void loop()
{
  digitalWrite(BrocheLED, HIGH);   // met la sortie num. à l'état haut (led allumée)
  delay(3000);                     // attente de 3 secondes
  digitalWrite(BrocheLED, LOW);    // met la sortie num. à l'état bas (led éteinte)
  delay(1000);                     // attente de 1 seconde
}
```


Le second exemple en changeant la résistance

- Que se passe-t-il si on réduit la résistance ?



Vous le verrez en TP !

Serial (ou comment lire et écrire)

- La librairie Serial est une librairie essentielle du langage Arduino qui permet de visualiser sur le PC des messages reçus depuis la carte Arduino ou de commander la carte Arduino.
- En couplant l'utilisation de cette librairie avec l'interface programmable graphique Processing côté PC, on dispose d'un outil extrêmement puissant pour réaliser toute sortes d'affichages graphiques sur le PC ou d'interactions entre la carte et le PC (commande de la carte Arduino avec la souris ou le clavier !).

Serial

- Elle est utilisée pour les communications par le port série entre la carte Arduino et un ordinateur ou d'autres composants. Toutes les cartes Arduino ont au moins un port Série (également désigné sous le nom de UART ou USART) : Serial. Ce port série communique sur les broches 0 (RX) et 1 (TX) avec l'ordinateur via le port USB.
  - C'est pourquoi, si vous utilisez cette fonctionnalité, vous ne pouvez utiliser les broches 0 et 1 en tant qu'entrées ou sorties numériques.
- Vous pouvez utiliser le terminal série intégré à l'environnement Arduino pour communiquer avec une carte Arduino. Il suffit pour cela de cliquer sur le bouton du moniteur série dans la barre d'outils puis de sélectionner le même débit de communication que celui utilisé dans l'appel de la fonction begin().
- La carte Arduino Mega dispose de trois ports série supplémentaires : Serial1 sur les broches 19 (RX) et 18 (TX), Serial2 sur les broches 17 (RX) et 16 (TX), Serial3 sur les broches 15 (RX) et 14 (TX).

Serial

- Les fonctions de la librairie
  - `begin()`
  - `available()`
  - `read()`
  - `flush()`
  - `print()`
  - `println()`
  - `write()`

Le baud

- Le baud est une unité de mesure utilisée dans le domaine des télécommunications en général, et dans le domaine informatique en particulier. Le baud est l'unité de mesure du nombre de symboles transmissibles par seconde.

- Le terme « baud » provient du patronyme d'Émile Baudot, l'inventeur du code Baudot utilisé en télégraphie.

- Il ne faut pas confondre le baud avec le bps (bit par seconde), ce dernier étant l'unité de mesure du nombre d'information effectivement transmise par seconde. Il est en effet souvent possible de transmettre plusieurs bits par symbole. La mesure en bps de la vitesse de transmission est alors supérieure à la mesure en baud.

Serial.begin()

- Description
  - Fixe le débit de communication en nombre de caractères par seconde (l'unité est le baud) pour la communication série.
  - Pour communiquer avec l'ordinateur, utiliser l'un de ces débits : 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. Vous pouvez, également, spécifier d'autres débits - par exemple, pour communiquer sur les broches 0 et 1 avec un composant qui nécessite un débit particulier.
- Syntaxe
  - Serial.begin(debit);
- Paramètres
  - int debit: debit de communication en caractères par seconde (ou baud). Pour communiquer avec l'ordinateur, utiliser l'un de ces débits : 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200.
- En pratique utiliser une valeur comprise entre 9600 et 115200. Typiquement 115200 fonctionne très bien ! (Plus le débit est élevé et plus la communication est rapide...)
  - En cas d'utilisation du terminal série, IL FAUDRA FIXER LE MEME DEBIT dans la fenêtre du Terminal !

Serial.available()

- Description
  - Donne le nombre d'octets (caractères) disponible pour lecture dans la file d'attente (buffer) du port série.
- Syntaxe
  - Serial.available();
- Valeur renvoyée
  - Le nombre d'octet disponible pour lecture dans la file d'attente (buffer) du port série, ou 0 si aucun caractère n'est disponible. Si une donnée est arrivée, Serial.available() sera supérieur à 0. La file d'attente du buffer peut recevoir jusqu'à 128 octets.

Serial.read()

- Description
  - Lit les données entrantes sur le port Série.
- Syntaxe
  - Serial.read();
- Valeur renvoyée
  - Renvoi le premier octet de donnée entrant disponible dans le buffer du port série, ou -1 si aucune donnée n'est disponible. (int)

Serial.write()

- Description
  - Écrit des données binaires sur le port série. Ces données sont envoyées comme une série d'octet; pour envoyer les caractères correspondants aux chiffres d'un nombre, utiliser plutôt la fonction print().
- Syntaxe
  - serial.write(val)
  - serial.write(str)
  - serial.write(buf, len)
- Paramètres
  - val: une valeur à envoyer sous forme d'octet simple
  - str: une chaîne à envoyer sous forme d'une série d'octets
  - buf: un tableau pour envoyer une série d'octets
  - len: la largeur du tableau
- Valeur renvoyée
  - Un octet représentant le nombre d'octets écrits (même si il est optionnel de récupérer cette valeur)

Serial.print()

- Description
  - Affiche les données sur le port série sous forme lisible pour les humains (texte ASCII). Cette instruction peut prendre plusieurs formes.
    - Les nombres entiers sont affichés en utilisant les caractères ASCII pour chaque chiffre.
    - Les nombres à virgules (float) sont affichés de la même façon sous forme de caractères ASCII pour chaque chiffre, par défaut avec 2 décimales derrière la virgule.
    - Les valeurs de type byte sont affichés sous la forme d'un caractère ASCII.
    - Les caractères et les chaînes sont affichés tels quels.
- Syntaxe

  Serial.print(val) ou Serial.print(val, format)
- Paramètre
  - val: la valeur à afficher. N'importe quel type de données.
  - format (optionnel) : spécifie
    - la base utilisée pour les nombres entiers
      - Les valeurs autorisées sont BIN (binaire, ou base 2), OCT (octal, ou base 8), DEC (décimal, ou base 10), HEX (hexadécimal, ou base 16) et plus ~~BYTE~~
    - le nombre de décimales (pour les nombres de type float)
- Valeur renvoyée
  - Un size_t (long) représentant le nombre d'octets écrits (même si il est optionnel de récupérer cette valeur)

Exemples

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

~~Serial.print(78, BYTE);~~ // N'est plus possible depuis la version 1.0
                           // Il faut utiliser Serial.write(78) qui affiche "N"
Serial.print(78, BIN) ; // affiche  "1001110"
Serial.print(78, OCT); // affiche "116"
Serial.print(78, DEC); // affiche "78"
Serial.print(78, HEX); // affiche "4E"
Serial.print(1.23456, 0); // affiche "1"
Serial.print(1.23456, 2); // affiche "1.23"
Serial.print(1.23456, 4); // affiche "1.2346"
```

Serial.println()

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

- Syntaxe

  Serial.println(val) ou Serial.println(val, format)

- Paramètres
  - val: la valeur à afficher. N'importe quel type de données.
  - format (optionnel) : spécifie
    - la base utilisée pour les nombres entiers
      - Les valeurs autorisées sont BIN (binaire, ou base 2), OCT (octal, ou base 8), DEC (décimal, ou base 10), HEX (hexadécimal, ou base 16) et plus ~~BYTE~~
    - le nombre de décimales (pour les nombres de type float)
- Valeur renvoyée
  - Un size_t (long) représentant le nombre d'octets écrits (même si il est optionnel de récupérer cette valeur)

Exemple

```
/*
 * Lit une valeur analogique en entrée sur analog 0 et affiche la valeur.
 */

int analogValue = 0;    // variable pour stocker la valeur analogique

void setup() {
  // initialise le port série à 9600 bauds
  Serial.begin(9600);
}

void loop() {
  // lit la valeur analogique sur la broche analog 0
  analogValue = analogRead(0);

  // affiche cette valeur dans différents formats
  Serial.println(analogValue);       // affichage décimal
  Serial.println(analogValue, DEC);  // affichage décimal
  Serial.println(analogValue, HEX);  // affichage hexadécimal
  Serial.println(analogValue, OCT);  // affichage octal
  Serial.println(analogValue, BIN);  // affichage binaire
  Serial.write(analogValue);         // affichage octet simple
  Serial.println();                  // affichage retour à la ligne

  // pause de 10 millisecondes avant la mesure suivante
  delay(10);
}
```

Serial.print() et Serial.println()

- Un problème majeur que l'on va rencontrer avec les chaînes de caractères, c'est l'utilisation de la RAM qui va rapidement saturer (2K pour une UNO contre 32K de mémoire programme) et donner des comportements inattendus. Pour contourner le problème :
  - il faut mettre les chaînes de caractères en mémoire programme, ce qui peut se faire simplement à l'aide de la librairie flash
  - on peut évaluer le niveau d'utilisation de la RAM en intégrant une fonction spécifique dans le programme. Voir : http://www.arduino.cc/playground/Code/AvailableMemory

Exemple de lecture et d'affichage sur le port série

```
int incomingByte = 0;   // variable pour lecture de l'octet entrant

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

void loop() {

        // envoie une donnée sur le port série seulement quand reçoit une donnée
        if (Serial.available() > 0) { // si données disponibles sur le port série
                // lit l'octet entrant
                incomingByte = Serial.read();

                // renvoie l'octet reçu
                Serial.print("Octet recu: ");
                Serial.println(incomingByte, DEC);
        }
}
```

Serial.flush()

- Description
  - Cette fonction s'assure que les données écrites sur le port série aient été physiquement envoyés avant de rendre la main (la fonction est donc bloquante).
- Syntaxe
  - Serial.flush();
- Ancienne description
  - Vide le buffer de réception de données du port série. Par conséquent, tout appel de la fonction Serial.read() ou Serial.avalaible() renverra seulement les données reçues après le plus récent appel de la fonction Serial.flush().

Mais ça c'était avant !

Bibliothèque pour la carte SD

- Cette librairie est très intéressante et ouvre des possibilités de stockage de données impressionnantes puisque les cartes SD disposent de 4 à 8 Go d'espace.
- De plus, les cartes SD une fois retirée de leur emplacement sur le shield Arduino, elles peuvent être lues directement sur un PC. En enregistrant les données dans un fichier texte au format CSV par exemple, on rend possible une utilisation immédiate des données dans un tableur avec réalisation de graphique, etc. Que du bon, et même du très bon !
- A noter que la librairie sdfatlib donne accès à davantage de fonctions, notamment des fonctions d'accès aux informations de la carte SD, ce qui peut être intéressant dans certains cas.
- Pour l'utiliser, il suffira de l'inclure et d'utiliser les sous programmes qu'elle propose :

```
#include <SD.h>
```

Quelques autres fonctions de Serial

- `if (Serial)`
- `end()`
- `find()`
- `findUntil()`
- `parseFloat()`
- `parseInt()`
- `peek()`
- `readBytes()`
- `readBytesUntil()`
- `setTimeout()`
- `serialEvent()`

Bibliothèque pour la carte SD

- INFORMATIONS SUR LES FICHIERS SUPPORTÉS
  - La librairie SD permet de lire et d'écrire les cartes mémoires SD, par exemple sur le shield Wireless SD.
  - La librairie supporte les systèmes de fichier FAT16 et FAT 32 sur les cartes mémoires SD standard et les cartes SDHC.
  - La librairie supporte l'ouverture de plusieurs fichiers à la fois et n'utilise toujours que les noms de fichiers courts au format "8.3" (c'est à dire des noms en 8 lettres maxi, un point et trois lettres, par exemple filename.txt).
  - Les noms de fichiers utilisés avec les fonctions de la librairie SD peuvent inclure des chemins séparés par des slash /, par exemple "directory/filename.txt).
  - Le répertoire de travail étant toujours la racine de la carte SD, un nom se réfère au même fichier, qu'il inclue ou non un / (par exemple, "/file.txt" est équivalent à "file.txt").

Bibliothèque pour la carte SD

- INFORMATIONS MATÉRIELLES
  - La communication entre la carte Arduino et la carte mémoire SD utilise la communication SPI, laquelle utilise les broches 11, 12 et 13 de la plupart des cartes Arduino ou les broches 50, 51 et 52 sur la carte Arduino Mega (à noter que la carte d'extension Arduino Wireless SD assure la connexion pour la carte Mega via le connecteur droit ICSP* de la carte Arduino).
  - En plus, une autre broche peut être utilisée pour sélectionner la carte SD. Cette broche peut être la broche matérielle SS (broche 10 sur la plupart des cartes Arduino et broche 53 sur la Mega) ou une autre broche spécifiée dans la fonction d'initialisation SD.begin().
  - Noter que même si vous n'utilisez pas la broche matérielle SS, elle doit être laissée en sortie, sinon la librairie ne fonctionnera pas.

*In Circuit Serial Programming

Bibliothèque pour la carte SD

- La classe `File` permet de lire et d'écrire dans les fichiers individuels sur la carte SD.
  - `available()`
  - `close()`
  - `flush()`
  - `peek()`
  - `position()`
  - `print()`
  - `println()`
  - `seek()`
  - `size()`
  - `read()`
  - `write()`
  - `isDirectory()`
  - `openNextFile()`
  - `rewindDirectory()`

Bibliothèque pour la carte SD

- La classe `SD` fournit les fonctions pour accéder à la carte SD et manipuler ses fichiers et répertoires.
- L'objet `SD` est directement disponible à l'inclusion de la librairie.
  - `begin()`
  - `exists()`
  - `mkdir()`
  - `open()`
  - `remove()`
  - `rmdir()`

SD.begin()

- Description
  - Initialise la librairie SD et la carte SD. Cela lance l'utilisation du bus SPI (broches numériques 11,12 et 13 sur la plupart des cartes Arduino) et initialise la broche de sélection du circuit intégré (chip select), qui est par défaut la broche matérielle SS (broche 10 sur la plupart des cartes Arduino).
  - Notez que même si vous utilisez une broche de sélection différente, la broche matérielle SS doit être laissée en SORTIE, sinon les fonctions de la librairie SD ne fonctionneront pas.

SD.begin()

- Syntaxe
  - SD.begin()

    ou
  - SD.begin(cspin)
- Paramètres
  - SD : l'objet racine de la librairie SD
  - cspin (optionnel): la broche connectée à la broche de sélection "chip select" de la carte SD. Cette broche est la broche matérielle SS du bus SPI par défaut.
- Valeur renvoyée
  - true si réussite;
  - false si échec.

SD.begin() : exemple

```
#include <SD.h>


void setup()
{
  Serial.begin(115200); // utiliser le meme debit coté Terminal Serie
  Serial.println("Initialisation de la SD card...");

  pinMode(10, OUTPUT); // laisser la broche SS en sortie - obligatoire avec librairie SD


  if (!SD.begin(10)) { // si initialisation avec broche 10 en tant que CS n'est pas réussie
    Serial.println("Echec initialisation!"); // message port Série
    return; // sort de setup()
  }
  Serial.println("Initialisation reussie !"); // message port Série

}

void loop(){

}
```

SD.exist()

- Description
  - Cette fonction teste si un fichier ou un répertoire existe sur la carte mémoire SD.
- Syntaxe
  - SD.exists(filename)
- Paramètres
  - SD : l'objet racine de la librairie SD
  - filename : le nom du fichier dont il faut tester l'existence, lequel peut inclure des répertoires (délimités par les slash /).
- Valeur renvoyée
  - true : si le fichier ou répertoire existe
  - false : si le fichier ou le répertoire n'existe pas

SD.mkdir()

- Description
  - Cette fonction crée un répertoire sur la carte mémoire SD. Cette fonction créera également les répertoires intermédiaires qui n'existe pas encore; par exemple :

    SD.mkdir("a/b/c");
  - créera les répertoires a, puis b dans a et c dans b.
- Syntaxe

  SD.mkdir(filename)
- Paramètres
  - SD : l'objet racine de la librairie SD
  - filename : le nom du répertoire ou chemin à créer - chaîne de caractères
- Valeur renvoyée
  - true : si la création du chemin a réussi
  - false : si la création du chemin a échoué

SD.open()

- Description
  - Cette fonction ouvre un fichier sur la carte mémoire SD. Si le fichier est ouvert pour écriture, il sera créé si il n'existe pas déjà (cependant le chemin le contenant doit déjà exister).
- Syntaxe

  SD.open(filename) ou SD.open(filename, mode)
- Paramètres
  - SD : l'objet racine de la librairie SD
  - filename : le nom du fichier à ouvrir, qui peut inclure le chemin (délimité par les slash /) - pointeur caractères char *
  - mode (optionnel): le mode d'ouverture du fichier, par défaut FILE_READ - byte. Les paramètres possibles sont :
    - FILE_READ: ouvre le fichier pour lecture, en démarrant au début du fichier.
    - FILE_WRITE: ouvre le fichier pour lecture et écriture, en démarrant à la fin du fichier.
- Valeur renvoyée
  - un objet File qui se réfère au fichier ouvert; si le fichier n'a pas pu être ouvert, cet objet sera évalué comme false dans une situation "booléenne", autrement dit, il est possible de tester la valeur renvoyée avec if (file).

SD.rmdir()

- Description
  - Cette fonction efface un répertoire (remove dir) de la carte mémoire SD. Le répertoire doit être vide.
- Syntaxe
  - SD.rmdir(filepath)
- Paramètres
  - SD : l'objet racine de la librairie SD
  - filepath : le nom du répertoire à effacer, incluant les sous-répertoires délimités par des slashes /.
- Valeur renvoyée
  - true : si le répertoire a bien été effacé
  - false : si le répertoire n'a pas pu être effacé.
  - Si le répertoire n'existe pas, la valeur renvoyée n'est pas spécifiée.

SD.remove()

- Description
  - Cette fonction efface un fichier de la carte mémoire SD.
- Syntaxe
  - SD.remove(filename)
- Paramètres
  - SD : l'objet racine de la librairie SD
  - filename : le nom du fichier, qui peut inclure le chemin délimité par des slashs /.
- Valeur renvoyée
  - true : si le fichier a bien été effacé
  - false : si le fichier n'a pas pu être effacé.
  - A noter que la valeur renvoyée n'est pas spécifiée si le fichier n'existe pas

Les fichiers

- On déclare une variable de type fichier :
  - File nomFile;
- On ouvre un fichier ainsi :
  - File f = SD.open("data.txt", FILE_WRITE);

file.close()

- Description
  - Cette fonction ferme le fichier, et s'assure que toute donnée écrite dans ce fichier est physiquement enregistrée sur la carte mémoire SD.
- Syntaxe
  - file.close()
- Paramètres
  - file : une instance de l'objet File (renvoyée par la fonction SD.open())
- Valeur renvoyée
  - Aucune valeur

file.peek()

- Description
  - Cette fonction lit un octet dans un fichier sans avancer au suivant. Ainsi, des appels successifs de la fonction peek() renverront la même valeur, jusqu'au prochain appel de la fonction read().
- Syntaxe
  - file.peek()
- Paramètres
  - file : une instance de l'objet File (renvoyée par la fonction SD.open())
- Valeur renvoyée
  - l'octet suivant (ou caractère)
  - ou -1 si aucun n'est disponible

file.read()

- Description
  - Cette fonction lit un octet dans un fichier.
- Syntaxe
  - file.read()
- Paramètres
  - file : une instance de l'objet File (renvoyée par la fonction SD.open())
- Valeur renvoyée
  - l'octet (ou caractère) suivant
  - -1 si rien n'est disponible

file.write()

- Description
  - Cette fonction écrit des données dans un fichier.
- Syntaxe
  - file.write(data)
  - file.write(buf, len)
- Paramètres
  - file : une instance de l'objet File (renvoyée par la fonction SD.open())
  - data : l'octet(byte), le caractère(char) ou la chaîne de caractère (char *) à écrire dans le fichier
  - len : le nombre d'éléments contenus dans buf
- Valeur renvoyée
  - Un octet représentant le nombre d'octet écrits (même si il est optionnel de récupérer cette valeur)

file.print()

- Description
  - Cette fonction écrit des données dans un fichier, lequel doit être ouvert pour écriture. Cette fonction affiche les nombres comme une séquence de chiffres, chacun comme un caractère ASCII (par exemple le nombre 123 est écrit sous la forme de 3 caractères '1', '2', '3').
- Syntaxe
  - file.print(data)
  - file.print(data, BASE)
- Paramètres
  - file : une instance de l'objet File (renvoyée par la fonction SD.open())
  - data: les données à écrire dans le fichier - type possibles : char, byte, int, long, ou string
  - BASE (optionnel): la base dans laquelle écrire les nombres : BIN pour binaire (base 2), DEC pour décimal (base 10), OCT pour octal (base 8), HEX pour hexadécimal (base 16). La base par défaut est la base décimale.
- Valeur renvoyée
  - Un octet représentant le nombre d'octets écrits (même si il est optionnel de récupérer cette valeur)

file.println()

Cette fonction est identique à *file*.println(), elle écrit des données **suivies** d'un saut de ligne (retour chariot + nouvelle ligne)

file.flush()

- Description
  - Cette fonction s'assure que les données écrites dans un fichier ont été physiquement enregistrées sur la carte mémoire SD. Ceci est fait automatiquement également lorsque le fichier est fermé avec la fonction close()
- Syntaxe
  - file.flush()
- Paramètres
  - file : une instance de l'objet File (renvoyée par la fonction SD.open())

file.size()

- Description
  - Cette fonction permet de connaître la taille d'un fichier (en nombre d'octets).
- Syntaxe
  - file.size()
- Paramètres
  - file : une instance de l'objet File (renvoyée par la fonction SD.open())
- Valeur renvoyée
  - la taille du fichier en nombre d'octets - type unsigned long

file.available()

- Description
  - Cette fonction vérifie si des octets sont disponibles en lecture dans le fichier.
- Syntaxe
  - file.available()
- Paramètres
  - file : une instance de l'objet File (renvoyée par la fonction SD.open())
- Valeur renvoyée
  - le nombre d'octets disponibles - int

file.seek()

- Description
  - Cette fonction permet de se placer à une nouvelle position, qui doit être comprise entre 0 et la taille du fichier (inclusif).
- Syntaxe
  - file.seek(pos)
- Paramètres
  - file : une instance de l'objet File (renvoyée par la fonction SD.open())
  - pos : la position à laquelle se placer - type unsigned long
- Valeur renvoyée
  - true : en cas de réussite
  - false : en cas d'échec.

file.position()

- Description
  - Cette fonction renvoie la position courante à l'intérieur du fichier, c'est-à-dire la position à laquelle le prochain octet sera lu ou écrit).
- Syntaxe
  - file.position()
- Paramètres
  - file : une instance de l'objet File (renvoyée par la fonction SD.open())
- Valeur renvoyée
  - la valeur de la position à l'intérieur du fichier - type long

file.isDirectory()

- Description
  - Les répertoires (ou dossiers) sont une sorte particulière de fichiers : cette fonction permet de savoir si le fichier courant est un répertoire ou non.
- Syntaxe
  - file.isDirectory()
- Paramètres
  - file : une instance de l'objet File (renvoyée par la fonction SD.open())
- Valeur renvoyée
  - Renvoie un boolean : true si le fichier courant est un répertoire, false sinon.

file.openNextFile()

- Description
  - Renvoie le fichier ou dossier suivant dans un répertoire.
- Syntaxe
  - file.openNextFile()
- Paramètres
  - file : une instance de l'objet File (renvoyée par la fonction SD.open()) qui ici représente un répertoire
- Valeur renvoyée
  - char : le nom du fichier ou répertoire suivant.

Exemple : liste le contenu

```
#include <SD.h>
File root;
void setup()
{
  Serial.begin(9600);
  pinMode(10, OUTPUT);
  SD.begin(10);
  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) {
       // si pas de nouveau fichier
       // renvoie le premier fichier dans le répertoire
       dir.rewindDirectory();
       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 {
       // les fichiers ont une taille, pas les répertoires
       Serial.print("\t\t");
       Serial.println(entry.size(), DEC);
     }
   }
}
```

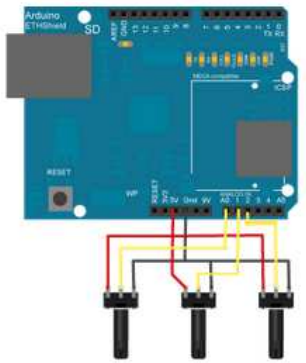
Un appel récursif

file.rewindDirectory()

- Description
  - Se place au niveau du premier fichier dans un répertoire. Utilisée en association avec openNextFile()
- Syntaxe
  - file.rewindDirectory()
- Paramètres
  - file : une instance de l'objet File (renvoyée par la fonction SD.open())

Un exemple de Datalogger sur SD

- 3 potentiomètres



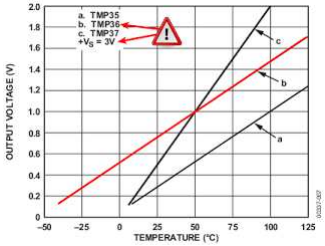
```
#include <SD.h>
const int chipSelect = 4;

void setup()
{
 // Open serial communications and wait for port to open:
  Serial.begin(9600);

  Serial.print("Initializing SD card...");
  // make sure that the default chip select pin is set to output, even if you don't use it:
  pinMode(10, OUTPUT); // 10 OU 4 ??? à CLARIFIER

  // 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");
  }
}
```

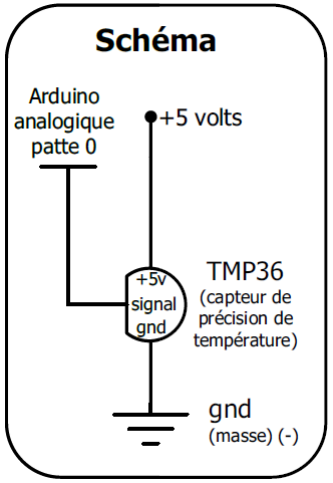
Mesure de la température

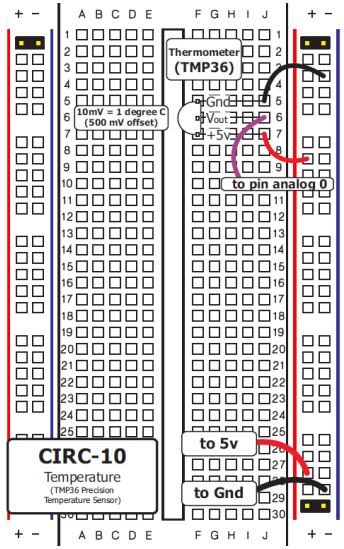
- Pour convertir ces valeurs en degrés, nous allons utiliser les capacités mathématiques de l'Arduino.
- Ensuite pour le visualiser nous allons utiliser un outil assez puissant fourni par le logiciel : la console de debug. Nous allons envoyer les valeurs par une connexion série pour les afficher à l'écran.
- Est-ce qu'on doit se poser des questions sur la fréquence d'échantillonnage ici ? Oui on peut se la poser, mais ça ne sert à rien vu la rapidité des variations de nos températures.
- Allez c'est parti !

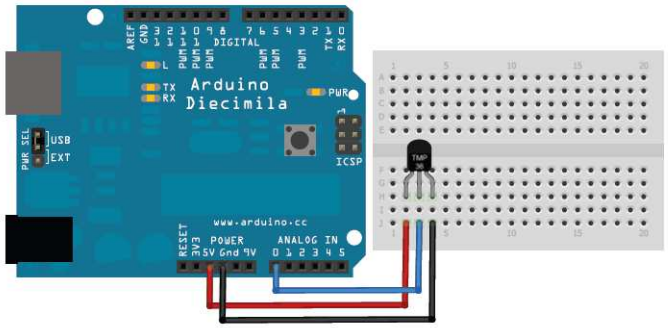
Mesure de la température

- Pour cela nous allons utiliser un CI (Circuit Intégré), le TMP36.
- Il a 3 pattes, la masse, le signal et le +5V, et est simple à utiliser.
- Il sort 10 millivolts par degré sur la patte de signal.
- Pour permettre de mesurer des températures négatives il y a un offset de 500mV
  - 25° C = 750 mV
  - 0° C = 500mV
  - -25° C = 250 mV)
- datasheet du TMP36 :
  http://ardx.org/TMP36










```
/* ---------------------------------------------
 * | Code d'Exemple du Kit d'Expérience Arduino |
 * | CIRC-10 .: Température :. |
 * ---------------------------------------------
 * Un programme simple pour afficher la température
 * sur la console du logiciel
 * Pour plus de détails sur ce circuit :
 */

//assignement des pattes du TMP36
int temperaturePin = 0; //la patte analogique à laquelle la patte Vout du TMP36 est connectée
                        //la resolution est de 10 mV / degré Celsius
                        //(500 mV d'offset) pour obtenir des températures négatives
void setup()
{
    Serial.begin(9600); //lance la connexion série avec l'ordinateur
                        //pour voir le résultat ouvrez la console série
                        //dernier bouton sous la barre de menu
                        //(il ressemble à rectangle avec une antenne)
}
void loop() //s'exécute à l'infini
{
    float temperature = getVoltage(temperaturePin); //obtiens la tension de sortie du
                                                    //capteur de temperature

    temperature = (temperature - .5) * 100; //conversion de 10mV
                                            //par degre avec 500mV d'offset en
                                            //degrés ((tension - 500mV) * 100)

    Serial.println(temperature); //affiche le résultat
    delay(1000); //attend une seconde
}

/*
 * getVoltage(int pin) – retourne la tension sur la patte analogique
 * définie par pin
 */
float getVoltage(int pin){
    return (analogRead(pin) * .004882814); //Convertie d'une plage de 0 à 1023
                                           //à une plage de 0 à 5V (Chaque pas égal environ 5 mV)
}
```

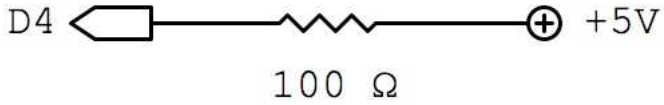
Arduino, une espèce à protéger

- Jusqu'à maintenant, nous nous sommes toujours contentés de faire circuler du courant du microcontrôleur à la masse (sauf pour les entrées analogiques).
- Au lieu d'utiliser les pattes du microcontrôleur seulement sous forme de sortie (OUTPUT), nous pouvons aussi les utiliser sous forme d'entrée (INPUT) ; c'est-à-dire qu'au lieu d'être raccordée à la masse, la patte sera raccordée au +5V.
  - Tout se passera bien, si au niveau du programme, la patte est configurée comme INPUT. Mais, si elle devait être configurée en OUTPUT et réglée à 0V (LOW) par erreur, il est presque certain que le microcontrôleur finira immédiatement au paradis des puces électroniques grillées.

Résistances de protection

Protection des entrées numériques

- Pour éviter de condamner notre microcontrôleur à la chaise électrique, nous allons devoir le protéger. Cela peut se faire en connectant sur la patte du microcontrôleur utilisé comme INPUT une résistance d'une centaine d'Ohms qui limitera le courant en cas de fausse manœuvre. Comme notre kit contient des résistances de 330 Ω, c'est celle que nous utiliserons.

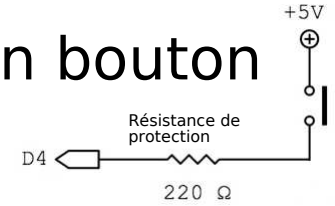


- Protéger votre Arduino doit devenir un réflexe. Si on doit mettre un INPUT, on protège avec une résistance au cas où on se trompe dans le code.

Résistances Pull-Up / Pull-Down

- Une résistance de rappel, aussi appelée résistance de tirage (en anglais pull-up resistor), est une résistance dans un circuit électronique, située entre la source d'alimentation et une ligne, et qui amène délibérément cette même ligne soit à l'état haut (1 logique) pour une résistance de rappel haut, soit à l'état bas (0 logique) pour une résistance de rappel bas.
- Et en clair ?

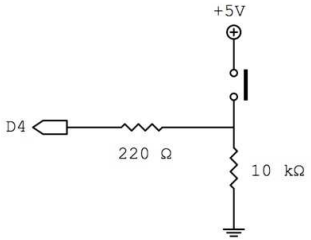
Observons l'état d'un bouton



- Modèle théorique :
  - Lorsque l'on presse le bouton poussoir, on envoie +5 Volts sur D4, qui sera interprété comme un 1.
  - Lorsqu'on relâche le bouton, il n'y a plus de courant qui arrive sur D4, ce qui sera interprété comme un 0. On a donc un signal binaire: allumé/éteint (on/off).
- Pourtant contrairement au cas théorique, fermer ou ouvrir un circuit (via un interrupteur ou un bouton poussoir) ne génère pas forcément un signal clair :
  - Le circuit non connecté à la source peut agir comme une antenne dans un environnement pollué d'ondes électromagnétiques (proches des moteurs par exemple). Cela va générer dans le circuit un courant qui peut être interprété comme un signal.
  - Il peut rester dans un circuit récemment ouvert un niveau d'énergie. Cela provoque des cas d'indétermination. Le signal qui a été coupé peut être considéré comme encore actif par un microcontrôleur.

Avec une résistance pull-down

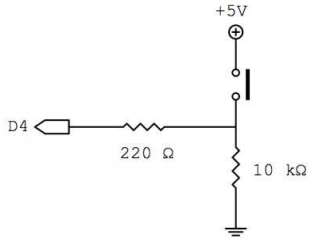
- Observons ce schéma :



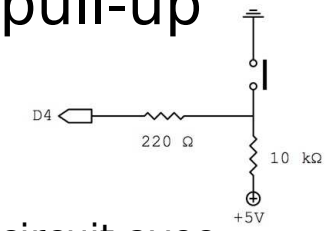
- Ce montage contient une résistance de 10 kΩ (soit 10000 Ω), qu'on appelle pull-down (littéralement tirer-en-bas). Cette résistance permet de tirer le potentiel vers le bas (pulldown). En français, on appelle aussi ceci un rappel au moins.
- **Le but d'une résistance de pull-down est donc d'évacuer les courants vagabonds et de donner un signal clair.**

Avec une résistance pull-down

- Si on presse le bouton, 5 Volts sont alors appliqués sur l'entrée. Le courant va prendre le chemin le plus simple, soit par la résistance de 220 Ω et finit par arriver sur D4. Si on relâche le bouton, la résistance pull-down ramène l'entrée à la masse (puisqu'elle est connectée à la masse).



Avec une résistance pull-up



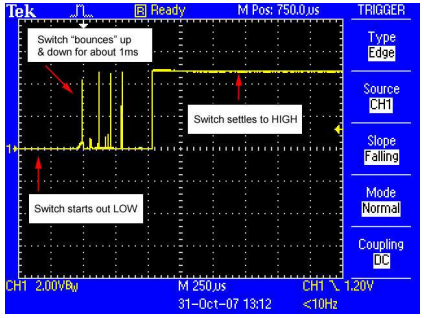
- Observons ce schéma :
- Si on le compare au schéma d'un circuit avec une résistance pull-down, on constate une inversion entre la terre et le +5V.
- En effet, lorsque le circuit est ouvert, une tension de +5V est appliquée à l'entrée D4.
- Lorsqu'on appuie sur le bouton poussoir, le courant électrique va passer par le chemin offrant le moins de résistance, soit directement par la masse (Gnd), sans passer par l'entrée D4.
- Le fonctionnement est donc inversé par rapport à la résistance pull-down.

Résistance pull-down ou pull-up ?

- À notre niveau, la seule chose qui va changer entre une résistance pull-down et une résistance pull-up est la lecture de l'information:
- Avec une résistance pull-down, par défaut, l'entrée sur la patte est égale à 0.
- Avec une résistance pull-up, par défaut, l'entrée sur la patte est égale à 1.
- Dans le code `if (digitalRead(bouton) == 1 )`, sera respectivement la valeur d'entrée lorsque le circuit est fermé (pull-down) ou ouvert (pull-up).
- Remarque :
  - La résistance est ici de 10KOhms, c'est une valeur assez commune permettant d'identifier facilement un circuit de pull-up/down.
  - Il est possible de diminuer cette résistances à 4.7 KOhms mais cela consommera aussi plus de courant.
  - Il n'est pas conseillé d'utiliser une résistance de pull-up de plus de 10KOhms. À partir de 100 KOhms, la résistance de pull-up/down interfère avec la circuiterie interne du micro-contrôleur et le résultat (la détection) deviendrait incertain.

On veut aller plus loin ?

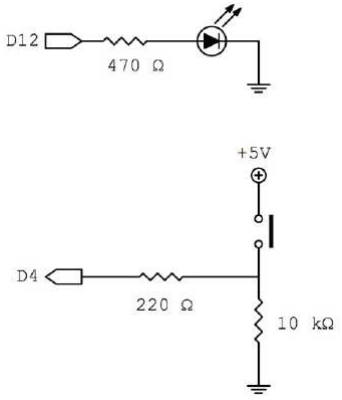
- Lorsque l'on presse/relâche un bouton, il y a souvent l'apparition de parasites pendant quelques millisecondes. Ces parasites n'apparaissent que durant les moments où l'on enfonce/relâche le bouton poussoir.

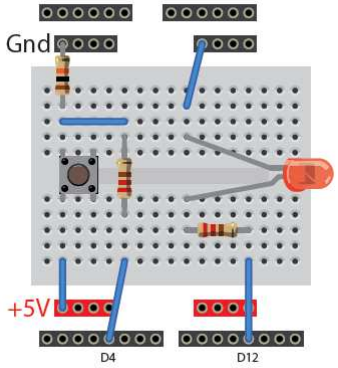


- Si l'on compte le nombre de pressions (pour faire un compteur), ces parasites viennent justement perturber le bon fonctionnement du logiciel. Le problème est matériel... et les parasites viennent ajouter des pressions fantômes.
- Il y a deux façon de corriger le problème :
  - Utiliser une capacité de déparasitage
  - Introduire un délai logiciel dans le programme.

Voir les références à la fin pour les détails

Montage avec résistance pull-down





Indiquez sur le schéma ci-dessus où se trouve la résistance pull down et la résistance de protection du microcontrôleur

Allume la LED quand on presse

```
/*
 * Allume LED en fonction de l'état du bouton poussoir
 */

const int bouton = 4;         // la patte 4 devient bouton
const int led = 12;           // la patte 12 devient led

void setup()
{
   pinMode(bouton, INPUT);     // Initialise la patte 4 comme entrée
   pinMode(led, OUTPUT);  // Initialise la patte 12 comme sortie
}

void loop()
{
   if (digitalRead(bouton) == 1 ) //teste si le bouton a une valeur de 1 (appuyé en pull-down)
   {
      digitalWrite(led, HIGH);       // allume la LED
   }
   else
   {
      digitalWrite(led, LOW);       // éteint la LED
   }
}
```

Si on échange, on a le comportement opposé. En électronique pur, il aurait fallu recâbler. On peut même envisager des effets (clignotements, fading, etc).

Les PIN et modes

- Les broches analogiques peuvent être utilisées en tant que broches numériques, représentées par les nombres 14 (entrée analogique 0) à 19 (entrée analogique 5).
- Chacune des broches numériques possède une résistance interne de pull-up de 20 à 50 KOhms déconnectée par défaut.
- La broche numérique 13 est plus difficile à utiliser que les autres en tant qu'entrée numérique car elle est associée à une résistance et sa LED soudées sur le circuit imprimé de la carte sur la plupart des cartes.
  - Si vous activez la résistance interne de rappel au plus de 20KOhms, cela mettra la broche à 1,7V au lieu des 5V théoriques car la LED et la résistance associées à la broche abaisse la tension, qui est toujours considérée au niveau BAS (LOW). Ainsi, si vous devez utiliser la broche 13 en tant qu'entrée numérique, utiliser une résistance de rappel au plus externe.

Allume la LED quand on presse

```
/*
 * Allume LED en fonction de l'état du bouton poussoir
 */

const int bouton = 4;         // la patte 4 devient bouton
const int led = 12;           // la patte 12 devient led
int etatbouton;               // variable qui enregistre l'état du bouton

void setup()
{
   pinMode(bouton, INPUT);     // Initialise le bouton comme entrée
   pinMode(led, OUTPUT);       // Initialise la led comme sortie
   etatbouton = LOW;           // Initialise l'état du bouton comme relâché
}

void loop()
{
   etatbouton = digitalRead(bouton); //On mémorise l'état du bouton

   if(etatbouton == LOW) //teste si le bouton a un niveau logique BAS
   {
      digitalWrite(led,LOW); //la LED reste éteinte
   }
   else  //teste si le bouton a un niveau logique différent de BAS (donc HAUT)
   {
      digitalWrite(led,HIGH); //le bouton est appuyé, la LED est allumée
   }
}
```

pinMode()

- Description
  - Configure la broche spécifiée pour qu'elle se comporte soit en entrée, soit en sortie.
- Syntaxe
  - pinMode(broche, mode)
- Paramètres
  - broche: le numéro de la broche de la carte Arduino dont le mode de fonctionnement (entrée ou sortie) doit être défini.
  - mode: soit INPUT (entrée en anglais) ou OUTPUT (sortie en anglais)

digitalWrite()

- Description
  - Met un niveau logique HIGH (HAUT en anglais) ou LOW (BAS en anglais) sur une broche numérique.
    - Si la broche a été configurée en SORTIE avec l'instruction pinMode(), sa tension est mise à la valeur correspondante : 5V (ou 3.3V sur les cartes Arduino 3.3V) pour le niveau HAUT, 0V (masse) pour le niveau BAS.
- Syntaxe
  - digitalWrite(broche, valeur)
- Paramètres
  - broche: le numéro de la broche de la carte Arduino
  - valeur : HIGH ou LOW (ou bien 1 ou 0)

digitalRead()

- Description
  - Lit l'état (= le niveau logique) d'une broche précise en entrée numérique, et renvoie la valeur HIGH (HAUT en anglais) ou LOW (BAS en anglais).
- Syntaxe
  - digitalRead(broche)
- Paramètres
  - broche : le numéro de la broche numérique que vous voulez lire. (int)
- Valeur retournée
  - Renvoie la valeur HIGH (HAUT en anglais) ou LOW (BAS en anglais)
- Remarque : Si la broche numérique en entrée n'est connectée à rien, l'instruction digitalRead() peut retourner aussi bien la valeur HIGH ou LOW (et cette valeur peut changer de façon aléatoire)

digitalWrite()

- Si la broche est configurée en ENTRÉE, écrire un niveau HAUT sur cette broche a pour effet d'activer la résistance interne de 20KOhms de "rappel au plus" (pullup) sur cette broche. À l'inverse, mettre un niveau BAS sur cette broche en ENTRÉE désactivera le pullup interne.
- Exemple :

```
pinMode(pin, INPUT);      // configure la broche en entrée
digitalWrite(pin, HIGH);  // écrit la valeur HIGH (=1) sur la broche en entrée
                          // ce qui active la résistance de "rappel au +" (pullup) au plus de la broche
```

- Cette instruction met la valeur 0/1 dans le bit de donnée qui est associé à chaque broche, ce qui explique qu'on puisse le mettre à 1, même si la broche est en entrée.

Exemple : on allume encore une LED avec un bouton mais avec le pull up interne !

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

void setup()
{
  pinMode(ledPin, OUTPUT);   // configure la broche 13 en SORTIE
  pinMode(inPin, INPUT);     // configure la broche 7 en ENTREE

  digitalWrite(inPin, HIGH); // écrit la valeur HIGH (=1) sur la broche en entrée
                             // ce qui active la résistance de "rappel au +"
                             // (pullup) au plus de la broche
}

void loop()
{
  val = digitalRead(inPin);   // lit l'état de la broche en entrée
                              // et met le résultat dans la variable

  digitalWrite(ledPin, val);    // met la LED dans l'état du BP
                                // (càd allumée si appuyé et inversement)
}
```

analogRead()

- Description
  - Lit la valeur de la tension présente sur la broche spécifiée. La carte Arduino comporte 6 voies (8 voies sur le Mini et le Nano, 16 sur le Méga) connectées à un convertisseur analogique-numérique 10 bits. Cela signifie qu'il est possible de transformer la tension d'entrée entre 0 et 5V en une valeur numérique entière comprise entre 0 et 1023. Il en résulte une résolution (écart entre 2 mesures) de : 5 volts / 1024 intervalles, autrement dit une précision de 0.0049 volts (4.9 mV) par intervalle !
  - Une conversion analogique-numérique dure environ 100 µs (100 microsecondes soit 0.0001 seconde) pour convertir l'entrée analogique, et donc la fréquence maximale de conversion est environ de 10 000 fois par seconde.

Exemple de lecture et d'envoi sur le Serial

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

int val = 0; // variable de type int pour stocker la valeur de la mesure

void setup()
{
    Serial.begin(9600);          //  initialisation de la connexion série
    // IMPORTANT : la fenêtre terminal côté PC doit être réglée sur la même valeur
}

void loop()
{

    // lit la valeur de la tension analogique présente sur la broche
    val = analogRead(analogPin);

    // affiche la valeur (comprise en 0 et 1023) dans la fenêtre terminal PC
    Serial.println(val);

}
```

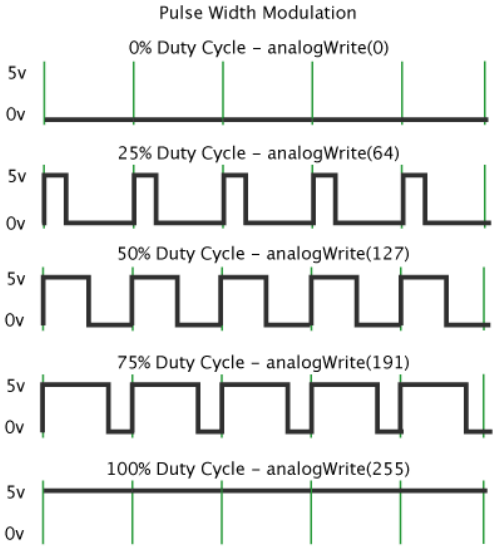
analogRead()

- Syntaxe
  - analogRead(broche_analogique)
- Paramètres
  - broche_analogique : le numéro de la broche analogique (et non le numéro de la broche numérique) sur laquelle il faut convertir la tension analogique appliquée (0 à 5 sur la plupart des cartes Arduino, 0 à 7 sur le Mini et le Nano, 0 à 15 sur le Mega)
- Valeur renvoyée
  - valeur int (0 to 1023) correspondant au résultat de la mesure effectuée
- Remarque :
  - Si la broche analogique est laissée non connectée, la valeur renvoyée par la fonction analogRead() va fluctuer en fonction de plusieurs facteurs (tels que la valeur des autres entrées analogiques, la proximité de votre main vis à vis de la carte Arduino, etc.).

analogWrite()

- Description
  - Génère une impulsion de largeur / période voulue sur une broche de la carte Arduino (onde PWM - Pulse Width Modulation en anglais ou MLI - Modulation de Largeur d'Impulsion en français).
    - Peut-être utilisé pour faire briller une LED avec une luminosité variable ou contrôler un moteur à des vitesses variables.
  - Après avoir appelé l'instruction analogWrite(), la broche générera une onde carrée stable avec un "duty cycle" (fraction de la période où la broche est au niveau haut) de longueur spécifiée (en %), jusqu'à l'appel suivant de l'instruction analogWrite() (ou bien encore l'appel d'une instruction digitalRead() ou digitalWrite() sur la même broche). La fréquence de l'onde PWM est approximativement de 490 Hz (soit 490 périodes par seconde).

analogWrite()



- Remarque : Sur les nouvelles cartes Arduino (incluant le Mini et le BT) avec le microcontrôleur ATmega168, cette fonction fonctionne sur les broches 3,5, 6, 9, 10 et 11. Les cartes plus anciennes USB et série basée sur l'ATmega8 ne supporte l'instruction analogWrite() que sur les broches 9, 10 et 11.

Exemple

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

```
int ledPin = 9;      // LED connectée sur la broche 9
int analogPin = 3;   // le potentiomètre connecté sur la broche analogique 3
int val = 0;         // variable pour stocker la valeur de la tension lue

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

void loop()
{
    val = analogRead(analogPin);   // lit la tension présente sur la broche en entrée
    analogWrite(ledPin, val / 4);  // Résultat d'analogRead entre 0 to 1023,
                                   // résultat d'analogWrite entre 0 to 255
                                   // => division par 4 pour adaptation
}
```

analogWrite()

- Syntaxe
  - analogWrite(broche, valeur);
- Paramètres
  - Broche : la broche utilisée pour "écrire" l'impulsion. Cette broche devra être une broche ayant la fonction PWM, Par exemple, sur la UNO, ce pourra être une des broches 3, 5 ,6 ,9 ,10 ou 11.
  - valeur: la largeur du "duty cycle" (proportion de l'onde carrée qui est au niveau HAUT) : entre 0 (0% HAUT donc toujours au niveau BAS) et 255 (100% HAUT donc toujours au niveau HAUT).
- Notes
  - Il n'est pas nécessaire de faire appel à l'instruction pinMode() pour mettre la broche en sortie avant d'appeler la fonction analogWrite().
  - L'impulsion PWM générée sur les broches 5 et 6 pourront avoir des "duty cycle" plus long que prévu. La raison en est l'interaction avec les instructions millis() et delay(), qui partagent le même timer interne que celui utilisé pour générer l'impulsion de sortie PWM.

Quelques fonctions bien utiles

- Mathématiques
  - `min(x, y)`
  - `max(x, y)`
  - `abs(x)`
  - `constrain(x, a, b)`
  - `map(valeur, fromLow, fromHigh, toLow, toHigh)`
  - `pow(base, exposant)`
  - `sq(x)`
  - `sqrt(x)`
- Temps
  - `unsigned long millis()`
  - `unsigned long micros()`
  - `delay(ms)`
  - `delayMicroseconds(us)`

constrain(x, a, b)

- Description
  - Contraint un nombre à rester dans une fourchette précise.
- Syntaxe
  - constrain(x,a,b)
- Paramètres
  - x: le nombre qui doit être contraint, tout type de donnée.
  - a: la limite inférieure de la fourchette, tout type de donnée.
  - b: la limite supérieure de la fourchette, tout type de donnée.
- Valeur renvoyée
  - Renvoie x si x a une valeur comprise entre a et b
  - Renvoie a si x est inférieur à a
  - Renvoie b si x est plus grand que b

map(value, fromLow, fromHigh, toLow, toHigh)

- Description (suite)
  - La limite basse de chaque fourchette peut être supérieure ou inférieure à la limite haute, dès lors l'instruction map() peut être utilisée pour inverser l'ordre des valeurs, par exemple :
    - y = map(x, 1, 50, 50, 1);
      - y évolue en sens inverse de x (càd si x = 1, y=50 et inversement)
  - Cette instruction supporte également des valeurs négatives, tel que dans cet exemple :
    - y = map(x, 1, 50, 50, -100);
      - Cette utilisation est aussi valide et fonctionne normalement.
  - L'instruction map() utilise des valeurs entières qui, par définition, ne peuvent fournir les décimales, alors que les calculs le devraient. Aussi la partie décimale est tronquée, et les valeurs ne sont pas arrondies ou moyennées.

map(value, fromLow, fromHigh, toLow, toHigh)

- Description
  - Ré-étalonne un nombre d'une fourchette de valeur vers une autre fourchette.
    - Ainsi, une valeur basse source sera étalonnée en une valeur basse de destination, une valeur haute source sera étalonnée en une valeur haute de destination, une valeur entre les deux valeurs source sera étalonnée en une valeur entre les deux valeurs destinations, en respectant la proportionnalité.
    - Cette fonction est très utile pour effectuer des changements d'échelle automatiques.
  - Cette fonction ne contraint pas les valeurs à rester dans les limites indiquées, car les valeurs en dehors de la fourchette sont parfois attendues et utiles. L'instruction constrain() doit être utilisée également avant ou après cette fonction, si les limites de la fourchette utilisée doivent être respectées.

map(value, fromLow, fromHigh, toLow, toHigh)

- Syntaxe
  - map (valeur, limite_basse_source, limite_haute_source, limite_basse_destination, limite_haute_destination)
- Paramètres
  - valeur : le nombre à ré-étalonner
  - limite_basse_source: la valeur de la limite inférieure de la fourchette de départ
  - limite_haute_source: la valeur de la limite supérieure de la fourchette de départ
  - limite_basse_destination: la valeur de la limite inférieure de la fourchette de destination
  - limite_haute_destination: la valeur de la limite supérieure de la fourchette de destination
- Valeur renvoyée
  - La valeur ré-étalonnée

Exemple de l'utilisation de map

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

```
int ledPin = 9;      // LED connectée sur la broche 9
int analogPin = 3;   // le potentiomètre connecté sur la broche analogique 3
int val = 0;         // variable pour stocker la valeur de la tension lue

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

void loop()
{
    val = analogRead(analogPin);   // lit la tension présente sur la broche en entrée
    // Ré-étalonne la valeur entre 0 et 1023 sur une fourchette entre 0 et 255
    val = map(val, 0, 1023, 0, 255);
    analogWrite(ledPin, val);
}
```

delay(ms)

- Description
  - Réalise une pause dans l'exécution du programme pour la durée (en millisecondes) indiquée en paramètre. (Pour mémoire, il y a 1000 millisecondes dans une seconde !)
- Syntaxe
  - delay (ms);
- Paramètres
  - ms (unsigned long): le nombre de millisecondes que dure la pause

Pour les matheux

- Voici la fonction entière map :

```
long map(long x, long in_min, long in_max, long out_min, long out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

- Comme nous l'avons vu map est fonction très puissante et très utile en pratique puisqu'elle permet de réaliser une adaptation d'échelle de valeur (entre la valeur d'une conversion analogique numérique brute et la valeur réelle de la tension par exemple) en une seule ligne, là où plusieurs lignes de calcul seraient normalement nécessaires.

Avertissements concernant delay(ms)

- Bien qu'il soit facile de faire clignoter une LED avec l'instruction delay(), et que de nombreux programmes utilisent de courtes pauses pour de nombreuses tâches telles que la commutation, l'utilisation de l'instruction delay() a des inconvénients non négligeables.
  - Aucune lecture de valeur sur un capteur, opération mathématique ou manipulation de bits ne peut avoir lieu durant une instruction delay(), et en effet, cela fait stopper toute autre activité le temps de l'exécution de la pause.
  - Pour une autre approche du contrôle des temporisations, voir l'instruction millis() et le programme d'exemple de clignotement sans délai.
  - Les programmeurs avancés (vous ?) évitent habituellement d'utiliser l'instruction delay() pour des pauses supérieures à 10 millisecondes, à moins que le programme Arduino ne soit très simple.
- Certaines choses se font cependant même lorsque l'instruction delay() est exécutée sur le microcontrôleur ATmega, car l'instruction delay() ne désactive pas les interruptions.
  - Les communications série qui arrivent sur la broche RX sont prises en compte, la génération d'impulsion PWM (analogWrite) et l'état des broches stables sont maintenus, et les interruptions fonctionnent comme elles le doivent.

delayMicroseconds(us)

- Description
  - Stoppe le programme pendant la durée (en microsecondes) indiquée en paramètre. (Pour mémoire, il y a 1000 microsecondes dans une milliseconde, et un million de microsecondes dans un seconde).
  - Actuellement, la valeur la plus grande qui peut produire une pause précise est 16383. Ceci pourra changer dans les versions futures d'Arduino. Pour des délais plus longs que quelques milliers de microsecondes, vous devriez plutôt utiliser l'instruction delay().
- Syntaxe
  - delayMicroseconds(us)
- Paramètres
  - us: le nombre de microsecondes que doit durer la pause.

millis()

- Description
  - Renvoie le nombre de millisecondes depuis que la carte Arduino a commencé à exécuter le programme courant. Ce nombre débordera (c'est-à-dire sera remis à zéro) après 50 jours approximativement.
- Syntaxe
  - variable_unsigned_long=millis();
- Valeur retournée
  - Le nombre de millisecondes depuis que le programme courant a démarré. Renvoie un long non signée.

Avertissements concernant delayMicroseconds(us)

- Cette fonction fonctionne de façon très précise pour des valeurs de 3 microsecondes et plus. Nous ne pouvons pas assurer que l'instruction delayMicroseconds donnera un résultat correct pour des durées plus courtes.
- L'instruction delayMicroseconds(0) générera un délai beaucoup plus long qu'attendu (1020 µs environ), comme si on utilisait un nombre négatif comme paramètre.

Exemple

```
unsigned long time;

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

void loop(){
  Serial.print("Time: ");
  time = millis();
  //affiche sur le PC le temps depuis que le programme a démarré
  Serial.println(time);
  // pause d'une seconde afin de ne pas envoyer trop de données au PC
  delay(1000);
}
```

Faire clignoter sans « delay »

```
const int ledPin =  13;
int ledState = LOW;      // l'état de la LED
long previousMillis = 0; // stocke le temps à laquelle la LED a été changée

long interval = 1000;    // intervalle auquel on veut clignoter (en millisecondes)

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  // vérifie si c'est le moment de faire clignoter la LED. C'est à dire si
  // la différence entre le temps actuel et le temps du précédent clignotement
  // de la LED est supérieur à l'intervalle auquel on veut qu'elle clignote.
  unsigned long currentMillis = millis();

  if(currentMillis - previousMillis > interval) {
    // sauvegarde le temps du précédent changement d'état de la LED
    previousMillis = currentMillis;

    if (ledState == LOW)
      ledState = HIGH;
    else
      ledState = LOW;

    digitalWrite(ledPin, ledState);
  }
  // ici on peut mettre d'autres instructions pour faire du boulot
  // en « parallèle » si ce n'est pas le moment de faire clignoter
}
```

?

Quelques mots encore sur les entrées/sorties

- Vous vous êtes peut être rendu compte que les entrées/sorties vues sont non bloquantes contrairement au `Saisir()` vu en algorithmique ou au `cin <<` vu en C++ qui stoppe l'exécution jusqu'à ce qu'une valeur ait été saisie
  - C'est à dire que les entrées/sorties sont asynchrones (non-bloquantes)
  - Ainsi, par exemple, lors d'un appel à digitalRead(unPin), on récupère la valeur sur la broche unPin, soit HIGH, soit LOW.

micros()

- Description
  - Renvoie le nombre de microsecondes depuis que la carte Arduino a démarré le programme en cours. Ce nombre déborde (repasse à 0) après approximativement 70 minutes.

    Sur les cartes Arduino à 16Mhz (par exemple le Nano), cette fonction a une résolution de 4 microsecondes (ce qui veut dire que la valeur retournée est toujours un multiple de quatre, autrement dit que la fonction compte de quatre en quatre). Sur les cartes Arduino à 8Mhz (par exemple Lilypad), cette fonction a une résolution de 8 microsecondes.

- Syntaxe
  - variable_unsigned_long=micros();
- Valeur renvoyée
  - Le nombre de microsecondes écoulées depuis que le programme en cours a démarré, sous la forme d'un nombre de type unsigned long.

Problèmes d'entrées/sorties non bloquantes

- Il faut être conscient qu'on peut « rater » des états

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

void setup()
{
  pinMode(ledPin, OUTPUT);   // configure la broche 13 en SORTIE
  pinMode(inPin, INPUT);     // configure la broche 7 en ENTREE

  digitalWrite(inPin, HIGH); // écrit la valeur HIGH (=1) sur la broche en entrée
                             // ce qui active la résistance de "rappel au +"
                             // (pullup) au plus de la broche
}

void loop()
{
  val = digitalRead(inPin);   // lit l'état de la broche en entrée
                              // et met le résultat dans la variable

  digitalWrite(ledPin, val);    // met la LED dans l'état du BP
                                // (càd allumée si appuyé et inversement)
  // ici un processus très long ou complexe pouvant durer plusieurs secondes, minutes
  // implique qu'on rate les événements d'appui sur le bouton
  for (int i = 0; i < 100; i++)
     delay(10);
}
```

Mais comment faire ?

- Dans l'exemple précédent on « ratait » les appuis sur un bouton pendant le processus, mais l'information aurait pu venir d'ailleurs, un capteur ultrason sur un robot par exemple et si la tâche longue consiste à avancer (ou faire d'autres opérations) et bien pendant ce temps là on peut rencontrer un obstacle sans en être capable de récupérer l'information !
  - On parle ici de rouler, mais toute opération bloquante (comme un délai) peut être un handicap.
- Comment faire si on veut exécuter une action particulière quand une broche change d'état ?

Polling versus Interruption

- On peut utiliser une interruption matérielle (hardware) qui appellera une fonction associée lors du changement d'état sur une broche.
- Le processeur exécutera le programme principal jusqu'à ce qu'il soit interrompu à l'arrivée d'une interruption qui déclenchera le code de la fonction associée à cette interruption.
- Une fois le code de la fonction associée à l'interruption sera fini, la « main » sera rendu au programme principal à l'endroit où l'exécution s'était arrêtée.
  - Attention entre temps (c'est-à-dire durant l'exécution de la fonction associée à l'interruption) des variables ont pu changer de valeurs (il peut y avoir des problèmes de cohérence)

Polling versus Interruption

- On peut faire du Polling (attente active ou « scrutation »)
  - Par exemple dans le `loop` si on veut attendre que la broche inPin passe à un état LOW, on peut utiliser le code suivant :

```
int val;
do
{
   val = digitalRead(inPin);
} while (val!= LOW);

// ici du code à exécuter
...
```

- L'inconvénient est que l'on ne fait rien d'autre que lire, dans l'attente d'un changement, alors que dans le code qui suit cette attente, il y a peut être des tâches indépendantes de la valeur `val` qui pourraient être exécutées (mais en les exécutant, on risque de « rater » le changement d'état)
- On consomme beaucoup de cycles CPU pour rien

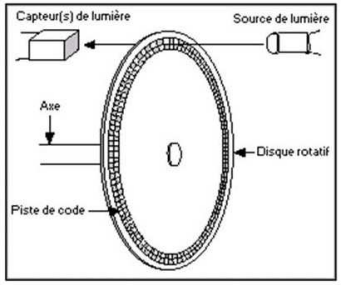
Exemple d'utilisation d'interruption

```
int ledPin = 13; // LED connectée à la broche n°13
int inPin = 2;   // un bouton poussoir connecté à la broche 2 car interruption 0
int INT0 = 0;
volatile int val = LOW;     // variable pour mémoriser la valeur lue

void setup()
{
  pinMode(ledPin, OUTPUT);   // configure la broche 13 en SORTIE

  digitalWrite(inPin, HIGH); // écrit la valeur HIGH (=1) sur la broche en entrée
                             // ce qui active la résistance de "rappel au +"
                             // (pullup) au plus de la broche
  // Attache la fonction à la patte ayant ce numéro d'interruption
  //et surveille tout CHANGEment d'état
  attachInterrupt(INT0, etatChange, CHANGE);
}

void loop()
{
  // ici un processus très long ou complexe pouvant durer plusieurs secondes, minutes
  // SANS qu'on rate les événements d'appui sur le bouton
  for (int i = 0; i < 100; i++)
     delay(10);
}

void etatChange()
{
  val = !val;   // l'état vient de changer, donc on le change dans val !

  digitalWrite(ledPin, val);    // met la LED dans l'état du BP
                                // (càd allumée si appuyé et inversement)
}
```

Avantages des interruptions

- On économise du CPU par rapport au polling
- On a un modèle de programmation événementiel permettant de simuler plusieurs tâches quand on ne dispose pas de threads.
- Elles sont très utiles pour faire des choses de façon automatique dans un programme pour microcontrôleur, et peuvent aider à résoudre les problème de temporisation. Une bonne utilisation pour une interruption peut-être la lecture d'un encodeur rotatif, pour contrôler les entrées de l'utilisateur.

Principe des souris à boule !



Opto-Interrupteur (polling)

```
const int OIPin = 2;   // Patte de l'Opto-interrupteur
const int ledPin = 13; // Patte de la LED

int beamState = LOW;   // état du faisceau

void setup() {

    pinMode(ledPin, OUTPUT); // initialise la patte de la LED comme sortie
    pinMode(OIPin, INPUT);   // initialise la patte de l'opto-interrupteur comme entrée
}

void loop() {
    beamState = digitalRead(OIPin); // lit l'état de l'opto-interrupteur

    // Vérifie si le faisceau a été interrompu
    // si il l'est, beamState est HIGH:
    if (beamState == HIGH) {
        digitalWrite(ledPin, HIGH); // allume la LED
    }
    else {
        digitalWrite(ledPin, LOW); // éteind la LED
    }
}
```

Aussi appelé opto-coupleur

Quelques cas d'utilisation

- Encodeur rotatif :
  - Si vous voulez que le programme prennent toujours en compte les impulsions venant d'un encodeur rotatif, sans jamais négliger une impulsion, il serait très compliqué d'écrire un programme pour faire quoique ce soit, parce que le programme devrait en permanence examiner les broches connectées à l'encodeur, afin de prendre en compte les impulsions lorsqu'elle surviendront.
- D'autres capteurs ont un rôle d'interface dynamique semblable, tel que :
  - capteur de sons essayant détecter un bruit
  - capteur infra-rouge (photo-transistor) essayant de détecter un obstacle
- Dans toutes ces situations, l'utilisation des interruptions libère le microcontrôleur pour faire d'autres choses tant que l'événement attendu ne survient pas.

Opto-Interrupteur (interruption)

```
const int OIPin = 2;   // Patte de l'Opto-interrupteur
const int ledPin = 13; // Patte de la LED
int INT0 = 0;

volatile int beamState = LOW;   // état du faisceau

void setup() {

    pinMode(ledPin, OUTPUT); // initialise la patte de la LED comme sortie
    // Attache la fonction à la patte ayant ce numéro d'interruption
    //et surveille tout front montant (LOW->HIGH) : RISING
    attachInterrupt(INT0, faisceauCoupe, RISING);
}

void loop() {

}

void faisceauCoupe()
{
  beamState = !beamState;   // l'état vient de changer, donc on le change dans val !

  digitalWrite(ledPin, beamState); // met la LED dans l'état correspondant au faisceau
                                   // (càd allumée si il est coupé et inversement)
}
```

J'en ai marre d'être interrompu !

Allons plus loin avec l'Opto-Interrupteur (et les interruptions)

```
/* Ce programme compte la vitesse d'un moteur CC en nombre de tours par minute.
 * Ce programme utilise un optocoupleur en fourche et une interruption externe.
 * Le résultat est affiché dans la fenêtre Terminal du logiciel Arduino. */

volatile long comptageImpuls=0; // variable accessible dans la routine interruption externe 0

long timeRef=0; // variable pour temps de référence en millisecondes
long delai=1000; // variable pour délai de comptage en millisecondes

void setup(){
  Serial.begin(115200); // initialise connexion série à 115200 bauds
  attachInterrupt(0, gestionINT0, RISING); // attache l'interruption 0 à la fonction gestionINT0()
  timeRef=millis(); // initialisation de la référence du comptage
}

void loop(){
  if (millis()>(timeRef+delai)) { // si le delai de comptage est écoulé
    timeRef=timeRef+delai; // réinitialise le délai de comptage

    Serial.print("Nombre Impulsions par seconde =");
    Serial.println(comptageImpuls); // affiche impulsions/sec
    Serial.print("Nombre tours par seconde =");
    Serial.println(comptageImpuls/2); // affiche nombre tours/sec

    comptageImpuls=0; // RAZ comptage impulsions
  }
}

void gestionINT0() {// la fonction appelée par l'interruption externe n°0
  comptageImpuls=comptageImpuls+1; // incrémente comptage Impulsion
}
```

Est ce qu'il peut y avoir des problèmes ?

attachInterrupt (interruption, fonction, mode)

- Paramètres
  - interruption : le numéro de l'interruption (type int)
  - fonction: la fonction à appeler quand l'interruption survient; la fonction ne doit recevoir aucun paramètres et ne renvoie rien. Cette fonction est également appelée une routine de service d'interruption (ou ISR).
  - mode : définit la façon dont l'interruption externe doit être prise en compte. Quatre constantes ont des valeurs prédéfinies valables :
    - LOW : pour déclenchement de l'interruption lorsque la broche est au niveau BAS
    - CHANGE : pour déclenchement de l'interruption lorsque la broche change d'état BAS/HAUT
    - RISING : pour déclenchement de l'interruption lorsque la broche passe de l'état BAS vers HAUT (front montant)
    - FALLING : pour déclenchement de l'interruption lorsque la broche passe de l'état HAUT vers l'état BAS (front descendant)
    - HIGH : pour déclenchement de l'interruption lorsque la broche est au niveau HAUT

attachInterrupt (interruption, fonction, mode)

- Description
  - Spécifie la fonction à appeler lorsqu'une interruption externe survient. Remplace tout autre fonction qui était attaché à cette interruption.

    La plupart des cartes Arduino ont deux interruptions externes : interruption externe n°0 sur la broche numérique 2 et interruption externe n°1 sur la broche numérique 3. (La carte Arduino Mega en possède quatre de plus : interruption externe n°2 sur la broche 21, n°3 sur la broche 20, n°4 sur la broche 19 et n°5 sur la broche 18.)
- Syntaxe
  - attachInterrupt (interruption, fonction, mode)

Interruptions : Important

- Attention
  - À l'intérieur de la fonction attachée à l'interruption, la fonction delay ne fonctionne pas et la valeur renvoyée par millis ne s'incrémente pas. Les données séries reçues pendant l'exécution de la fonction sont perdues.
  - La fonction delayMicroseconds fonctionne normalement à l'intérieur de la fonction attachée.
  - Vous devrez déclarer en tant que volatile toute les variables que vous modifier à l'intérieur de la fonction attachée à l'interruption.

volatile

Oups ! Il est temps de finir ce cours !

volatile

- Le mot-clé `volatile` est connu en tant que qualificateur de variable et il est utilisé usuellement avant la déclaration d'une type de variable, afin de modifier la façon dont le compilateur, et par voie de conséquence le programme, traitera la variable.
- Déclarer une variable volatile est une information pour le compilateur. Le compilateur est la partie du logiciel Arduino qui transforme votre code C/C++ en un code machine (c'est à dire en une suite de 0 et 1), lequel correspond aux instructions réelles pour le microcontrôleur Atmega de votre carte Arduino.
- Plus précisément, il indique au compilateur de charger la variable depuis la RAM et pas depuis un registre de stockage, lequel est un emplacement temporaire de la mémoire où les variables sont stockées et manipulées. Sous certaines conditions, la valeur de la variable stockée dans les registres peut-être imprécise.
- Une variable devra être déclarée volatile chaque fois que sa valeur pourra être changée par quelque chose d'autre que le code dans laquelle elle apparaît, tel qu'un fil d'exécution concurrent. Avec Arduino, la seule situation où cela risque d'arriver concerne les sections de codes associées aux interruptions, appelées également routines de service des interruptions.

detachInterrupt(interruption)

- Description
  - Désactive l'interruption externe spécifiée
- Syntaxe
  - detachInterrupt(interruption)
- Paramètres
  - interruption: le numéro de l'interruption externe à désactiver.

noInterrupts()

- Description
  - Désactive les interruptions (vous pouvez les réactiver avec l'instruction interrupts()). Les interruptions permettent à certaines tâches importantes de survenir à tout moment, et elles sont activées par défaut avec Arduino. Plusieurs fonctions ne fonctionnent pas lorsque les interruptions sont désactivées et les communications sont ignorées (série en particulier). Les interruptions peuvent perturber le déroulement du code et peuvent être désactivées pour des sections critiques du code.
- Syntaxe
  - noInterrupts()

interrupts()

- Description
  - Réactive les interruptions (après qu'elles aient été désactivée par l'instruction noInterrupts().
- Syntaxe
  - interrupts()

Exemple

```
void setup() {}

void loop()
{
  noInterrupts(); // désactivation des interruptions

  // code critique, sensible au temps, ici

  interrupts(); // réactivation des interruptions

  // le reste du code ici

}
```

Les interruptions

- Nous venons de voir des exemples d'interruptions matérielles.
- Mais sachez qu'il existe aussi des interruptions logicielles (par exemple déclenchées par des timers)

Des choses beaucoup trop avancées pour vous

- La manipulation des ports via les registres de port :

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

La famille


Arduino Uno


Arduino Leonardo


Arduino Due

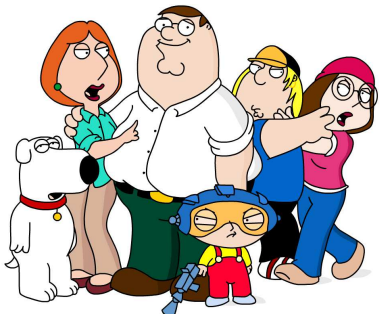

Arduino Esplora


Arduino Mega 2560


Arduino Mega ADK


Arduino Ethernet


Arduino Mini




LilyPad Arduino


LilyPad Arduino USB


Arduino Micro


Arduino Nano


Arduino Pro Mini


Arduino Pro

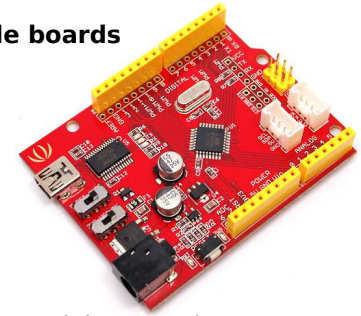

Arduino Fio

Des cousins plus ou moins proches

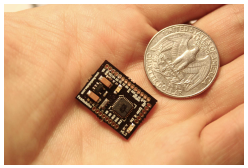
Arduino footprint-compatible boards


Freeduino 0603 SMT prototype


Scooby-Doo and his cousin Scooby-Dum.


Seeeduino V3.0 (Atmega 328P)

Software-compatibility only


Femtoduino Kit

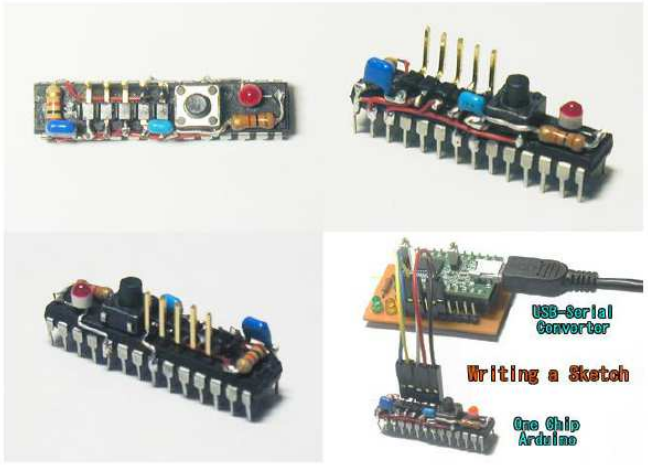

TinyDuino

En savoir plus http://en.wikipedia.org/wiki/List_of_Arduino_boards_and_compatible_systems

Des jolis Arduino

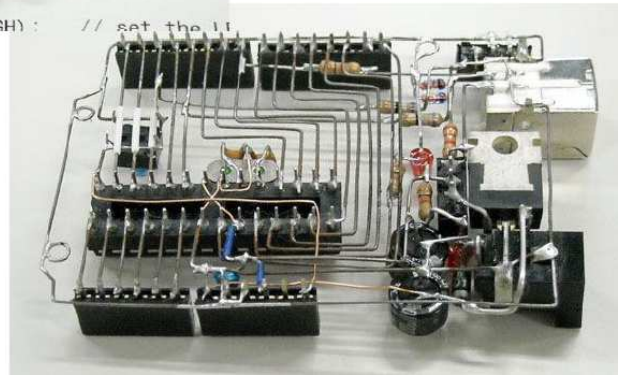
One chip Arduino

Kimio Kosaka



http://www.geocities.jp/arduino_diecimila/obaka/project-2/index_en.html

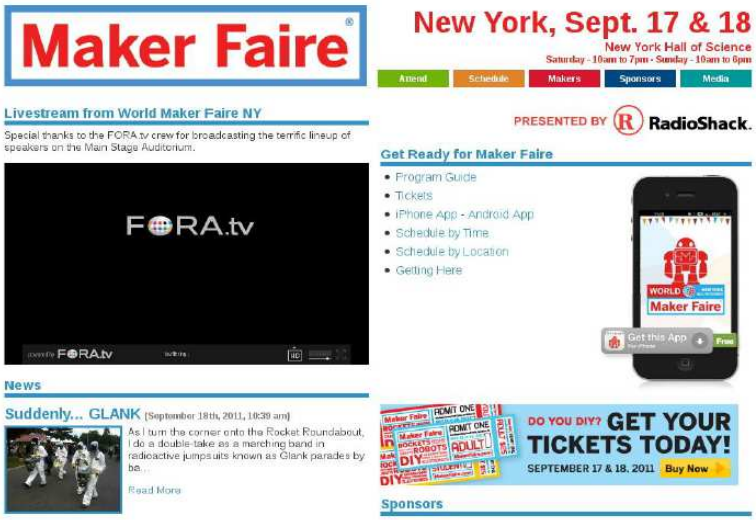
Des jolis Arduino



Apprendre soit même

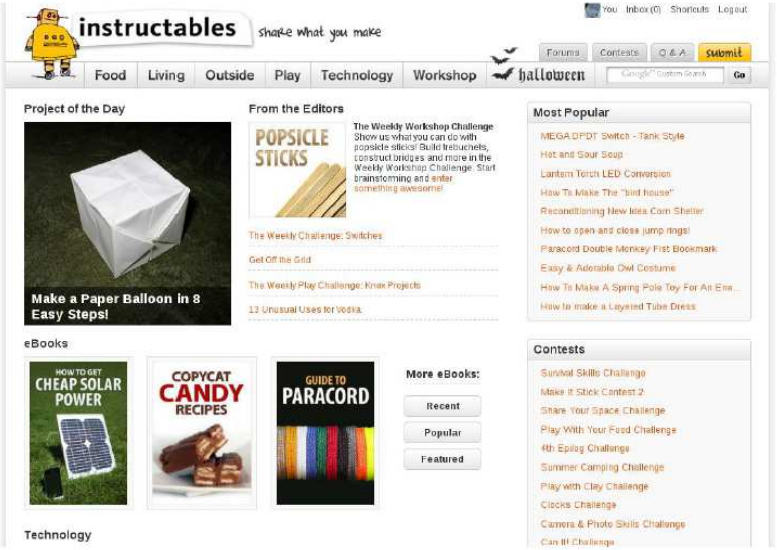
http://makezine.com/

Apprendre soit même



http://makerfaire.com/

Apprendre soit même



http://www.instructables.com/

Pour en savoir plus

- Le documentaire sur Arduino :
  http://www.archive.org/details/Arduino.TheDocumentary.English
- Arduino en BD :
  http://www.jodyculkin.com/wp-content/uploads/2011/09/arduino-comic-latest1.pdf
- Arduino (livre gratuit)
  http://fr.flossmanuals.net/_booki/arduino/arduino.pdf
- ADC et Arduino
  https://sites.google.com/site/measuringstuff/the-arduino
- En savoir plus sur les LED
  http://www.positron-libre.com/cours/electronique/diode/led/diode-led.php
- Pour une découverte ludique de l'électricité, je vous conseille de regarder le *C'est pas sorcier* consacré à ce thème
  - Partie 1: http://00.lc/cl
  - Partie 2: http://00.lc/cm

Pour en savoir plus

- Bouton poussoir (pull-up/pull-down et déparasitage)

  http://arduino103.blogspot.fr/2011/05/entree-bouton-resistance-pull-up-pull.html
- Asservissement en vitesse d'un moteur avec Arduino

  http://www.ferdinandpiette.com/blog/2012/04/asservissement-en-vitesse-dun-moteur-avec-arduino/
- Les 10 points essentiels pour choisir un numériseur/oscilloscope
  - Contient des choses sympas que je voulais mettre dans le cours

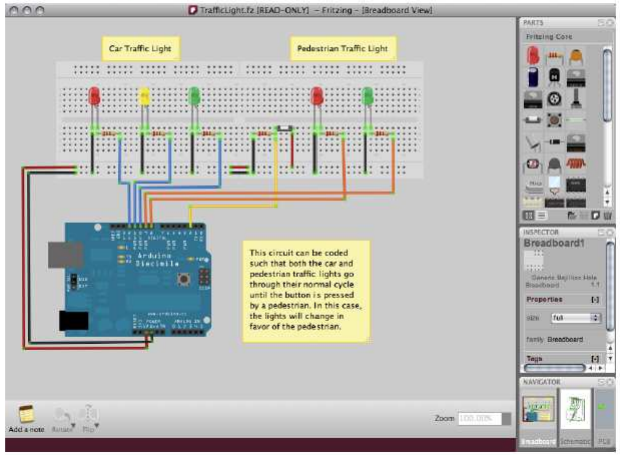
  http://www.ni.com/white-paper/4333/fr

Des ressources

- Web :
  - http://www.arduino.cc/
  - http://www.ladyada.net/learn/arduino/
  - http://www.todbot.com/blog/category/arduino/
  - http://www.freeduino.org/
  - http://www.adafruit.com/
  - http://www.sparkfun.com/
  - http://www.seeedstudio.com/
  - http://www.evola.fr/
- Livres :
  - "Arduino Programming Notebook", Brian W. Evans
  - "Physical Computing", Dan O'Sullivan & Tom Igoe
  - "Making Things Talk", Tom Igoe
  - "Hacking Roomba", TodE. Kurt

Des conseils

- Installez l'éditeur de montage Fritzing. Il vous sera utile pour garder en mémoire les montages réalisés. http://fritzing.org/
  - Pour dessiner des dessins de type breadboard, **des schémas**, des PCB (typon) en vue de réaliser des circuits.
  - Le site[11] regorge de schémas.



Electronique / électricité: recommandations

- **Attention** : entre 0v et 30v pas de risque, à 220V c'est mortel
- **Les manipulation à 110v ou 220v** : demande un minimum de connaissance et une rigueur stricte (demandez conseil à une personne expérimentée)
- Consultez les schémas et les programmes sur les sites[6] et[7] (très nombreux exemples)
- Pour des simulations de la partie électronique utilisez des simulateurs et expérimentez : icircuit[8] ou Ktechlab[9]

[6] http://www.arduino.cc/
[7] http://www.fritzing.org/
[8] http://www.falstad.com/circuit/
[9] http://sourceforge.net/projects/ktechlab/