

Mettre en pause les interruptions pour les codes critiques :

```
noInterrupts();  
    //Code that MUST be not interrupt  
interrupts();
```

Mathématiques et temps

Général

Sinus	$\sin(x,y)$	Cosinus	$\cos(x)$	Tang.	$\tan(x,y)$
Mini.	$\min(x,y)$	Racine	\sqrt{x}	Absolu	$\text{abs}(x,y)$
Maxi.	$\max(x,y)$	Puiss.	$\text{pow}(x,y)$		

constrain(x,min,max) : Retourne *x* si *min* < *x* < *max* sinon une valeur extrême.

map(x,minSource,maxSource,minDest,maxDest) : Retourne *x* adapté aux extrêmes de destination en fonction des extrêmes sources.

Aléatoire

Changer la *seed* de l'aléatoire :

```
randomSeed ( analogRead (0) );  
randomSeed ( numberValue );
```

Avoir un nombre aléatoire, *min* inclusif, *max* exclusif

```
long random (max);  
long random (min , max);
```

Temps

Temps écoulé depuis le démarrage :

```
unsigned long millis();  
unsigned long micros();
```

Overflow de *millis()* et *micros()* à 50 jours et 70 minutes.

Faire patienter quelques instants :

```
delay (milliSeconds);  
delayMicroseconds (microSeconds);
```

Mémoire

EEPROM

Un Arduino a une mémoire flash pour les variables, l'EEPROM. Attention, le nombre d'écriture est limitée !

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

Vider l'EEPROM :

```
for (int i = 0 ; i < EEPROM.length() ; i++) {  
    EEPROM.write(i, 0);  
}
```

Écrire dans l'EEPROM :

```
EEPROM.write (address , value); // 0-255  
EEPROM.put (address , value); // all
```

Lire dans l'EEPROM, le *get()* ne nécessite pas de cast :

```
byte value1;  MyObject value2;  
EEPROM.read (address , value1); // 0-255  
EEPROM.get (address , value2); // all
```

Mettre à jour une valeur que si besoin :

```
EEPROM.update (address , value); // 0-255
```

Autres

S'il manque de la mémoire pour une chaîne de caractères, la macro *F(string)* permet de la stocker dans la flash :

```
Serial.print(F("Very long string"));
```

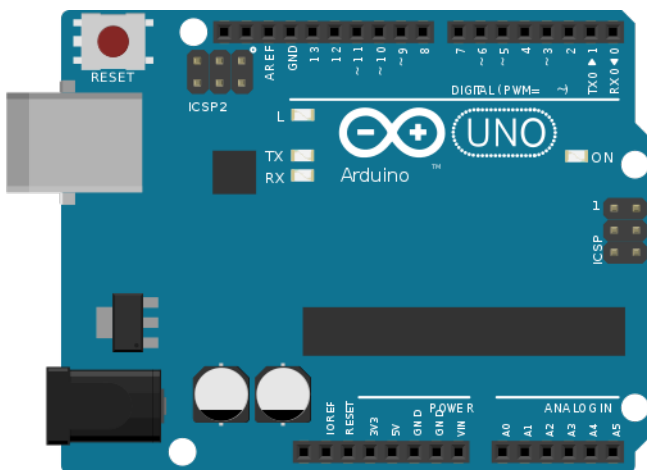
Taille d'une variable, un type, un tableau en bytes :

```
sizeof(int); int a = 1; sizeof(a);
```

Stocker dans la flash, à la place de la SRAM :

Suivant le compilateur, une des deux versions fonctionnent !

```
#include <avr/pgmspace.h>  
const type varName[] PROGMEM = {d0, d1, ...};  
const PROGMEM type varName[] = {d0, d1, ...};
```



<http://github.com/Servuc/jaizappe> \LaTeX Licence GPLv3

#Jaizappé le C++ Arduino

Histoire

L'Arduino, ou Genuino est une plateforme de prototypage basée sur un micro-contrôleur *Atmel* en général.

Il existe plusieurs boards avec chacune des spécificités.

Ce mémo est inspiré de la documentation Arduino.

Base

Code de base

Le code d'un projet Arduino de base :

```
void setup() {  
    //Exécuté en premier  
}  
void loop() {  
    //Exécuter après setup() en boucle  
}
```

Calcul binaire

Récupérer la partie *big* et *low* :

```
highByte (value);  
lowByte (value);
```

Lire, écrire les bits à partir de la droite, *value* : valeur numérique

```
bitRead (value , position);  
bitWrite (value , position , bit); // bit = 0 ou 1
```

Quelques raccourcis (*value* et *position*) :

```
bitSet (val , pos); // bitWrite (val , pos , 1);  
bitClear (val , pos); // bitWrite (val , pos , 0);
```

Analyse des caractères

```
isAlphaNumeric (thisChar); // [a-zA-Z0-9]  
isAlpha (thisChar); // [a-zA-Z]  
isAscii (thisChar);  
isWhitespace (thisChar);  
isControl (thisChar); // \n \r ...  
isDigit (thisChar); // [0-9]  
isLowerCase (thisChar); // [a-z]  
isUpperCase (thisChar); // [A-Z]  
isPrintable (thisChar); // Affichable  
isPunct (thisChar);  
isHexadecimalDigit (thisChar);
```

Débugage et série

Serial

Initialisation (une seule fois!) :

```
Serial.begin( SPEED );
```

SPEED vaut **9600**, **57600** ou **115200** en général. Vitesse en baud.

Écrire le moniteur de débog :

```
Serial.print( VARIABLE ); //Pas de \n
Serial.println( VARIABLE ); //Avec \n
```

Dans le cas d'un *float* :

```
Serial.print( VARIABLE, precision );
```

Lire ce qui est sur le port série (byte par byte), utiliser dans *loop()* :

```
if ( Serial.available() > 0 ) {
    byte incomingByte = Serial.read();
}
```

SoftwareSerial

Fonctionne comme *Serial*. Nécessite :

```
#include <SoftwareSerial.h>
```

Initialisation (inutile pour *Serial*) :

```
SoftwareSerial mSerial(PIN_RX, PIN_TX);
```

IMPORTANT! Pour communiquer entre 2 supports :

— Support 1 : RX ↔ TX : Support 2

— Support 1 : TX ↔ RX : Support 2

Si *SoftwareSerial* sont initialisés, un seul peut être en écoute :

```
mSerial.listen();
mSerial.isListening();
```

Les Pins

Il est conseillé de définir les pins :

```
#define PIN_FCT_ABCD 5
```

Indiquer le mode le pin (dans *setup()*) :

```
pinMode( PIN_ID, INPUT ); //Lecture
pinMode( PIN_ID, OUTPUT ); //Ecriture
```

Analogiques

Notées *AX* (*X* un nombre). **PIN_ID** noté *AX* aussi.

Mettre la valeur d'un pin (mode **OUTPUT**) :

```
analogWrite( PIN_ID, value ); // 0 – 255
```

Récupérer la valeur d'un pin (pas de mode) :

```
int myVal = analogRead( PIN_ID );
float myVoltage = myVal * (VOLT_MAX / 1023.0);
```

Fixé la valeur de lecture sur les pins analogiques :

```
analogReference( value );
```

Arduino	value	Description
Tous	DEFAULT	5V ou 3.3V
ATMega 168/328	INTERNAL	1.1V
ATMega 8	INTERNAL	2.56V
Ard. Mega	INTERNAL1V1	1.1V
Ard. Mega	INTERNAL2V56	2.56V
Tous	EXTERNAL	Sur <i>AREF</i> (0V – 5V)

Digitals

Notées *dX* ou *X* (*X* un nombre). Égale à **0** ou **1**.

Correspondant respectivement à **LOW** et **HIGH**.

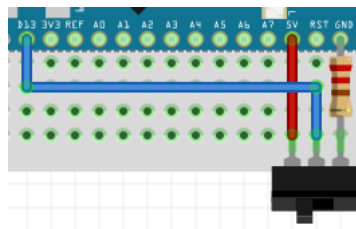
Mettre la valeur d'un pin (mode **OUTPUT**) :

```
digitalWrite( PIN_ID, HIGH ); //1 = 5V ou 3.3V
digitalWrite( PIN_ID, LOW ); //0 = 0V
```

Récupérer la valeur d'un pin (mode **INPUT**) :

```
int myVal = digitalRead( PIN_ID );
```

Cependant, il faut stabiliser le pin à 0 ou 5/3.3V. Exemple :



Usage avancé

Inutilisable avec les Arduino *Gemma* et *Due*.

Émettre une tonalité avec un Arduino avec un piezo, un haut parleur, **pin** doit être noté *PWM*. **time** en millisecondes.

Arduino	Fréq. min	Fréq. max
Tous	31	65535
Zero	41	275000

```
tone( pin, frequency ); //Ou
tone( pin, frequency, time );
```

Pour arrêter la première instruction :

```
noTone();
```

Lire une impulsion (signal continu) sur un pin digital de 10 microsecondes à 3 minutes,
value : **HIGH** et **LOW**, *timeout* : Attente max. en microsecondes.

```
pulseIn( pin, value );
pulseIn( pin, value, timeout );
```

Veille et interruptions

Pour mettre un Arduino :

```
delay( MILLI_SECONDS );
delayMicroseconds( MICRO_SECONDS );
```

Un *delay()* peut être arrêté par une interaction sur un pin digital :

Uno, Nano, base 328	2,3
Mega, Mega2560, MegaADK	2,3,18,19,20,21
Micro, Leonardo, base 32u4	0,1,2,3,7
Zero	Toutes sauf la 4
MKR1000 Rev.1	0,1,4,5,6,7,8,9,A1,A2
Due, 101	Toutes

Attacher une fonction à l'interruption :

```
attachInterrupt(
    digitalPinToInterrupt( PIN_ID ),
    FUNCTION, MODE);
```

- FUNCTION** : Une fonction de type *void* sans paramètre ;
- MODE** :
 - LOW** : Si le pin est à 0 (**LOW**) ;
 - CHANGE** : Tension qui change ;
 - RISING** : **LOW** à **HIGH** ;
 - FALLING** : **HIGH** à **LOW** ;
 - HIGH** : Pin à 1 (Sur *Due*, *Zero* et *MKR1000*).

Enlever une interruption sur un pin :

```
detachInterrupt(
    digitalPinToInterrupt( PIN_ID ));
```

IMPORTANT : Les variables partagées doivent être *volatile* :

```
volatile int mCount = 0;
void buttonPressed () { //Fct interrup.
    mCount++;
}
void loop() {
    Serial.println(mCount); delay(7331);
}
```