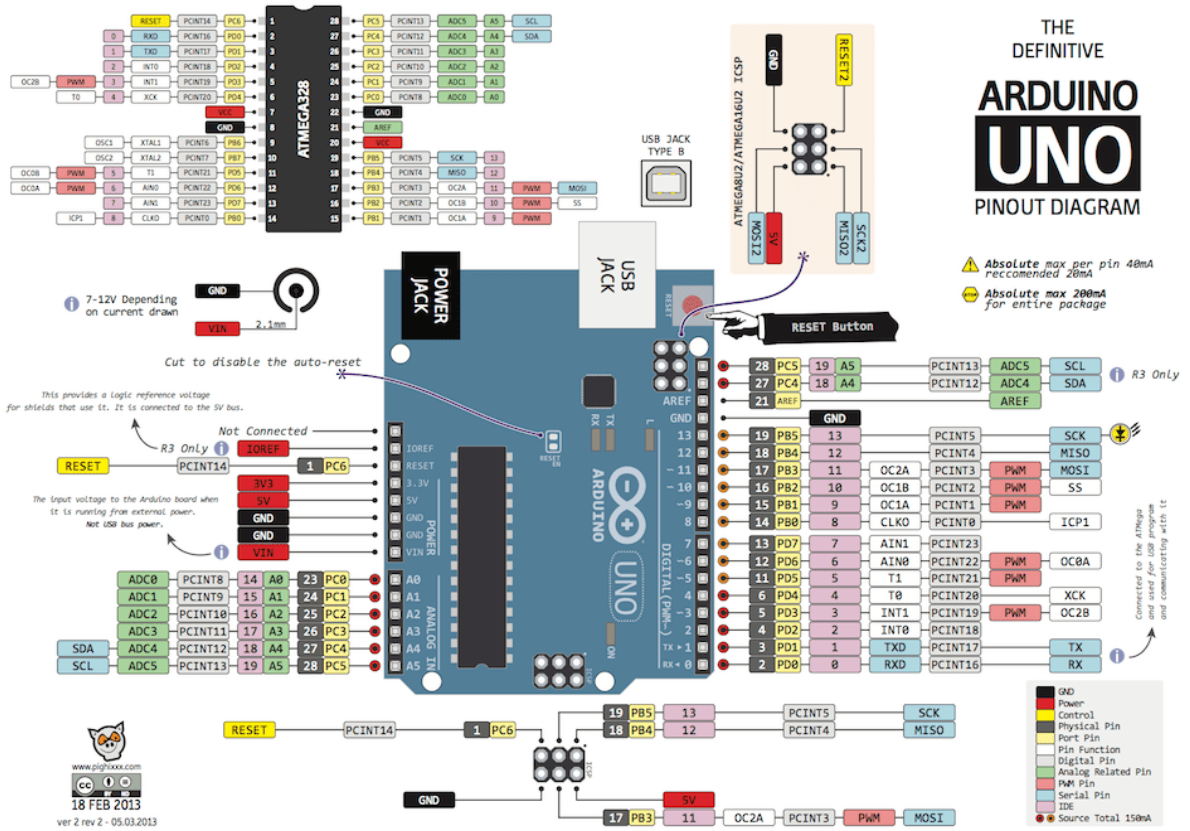
Prise en main d'un ARDUINO

Aide-mémoire pour des fonctionnalités de base

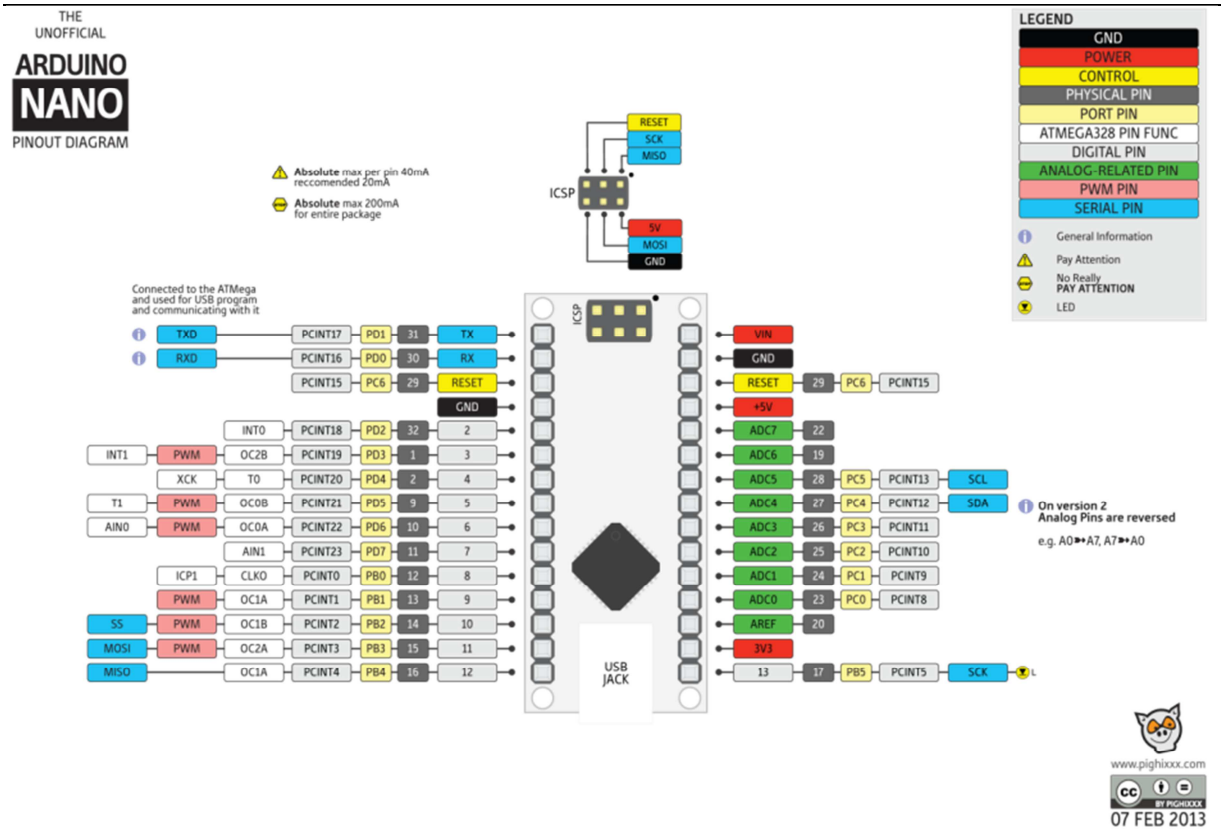
Sommaire

1 Carte ARDUINO

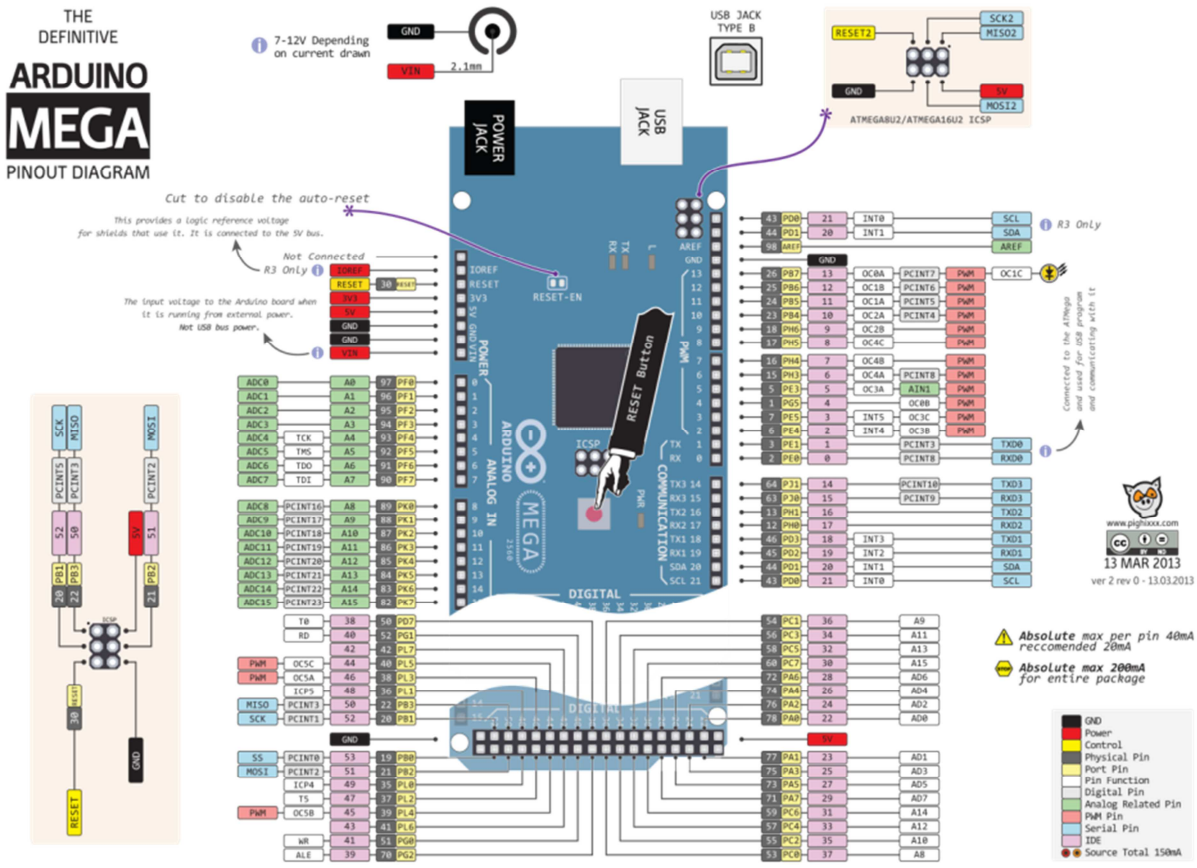
1.1 Bornes des cartes UNO, NANO et MEGA



Arduino UNO *(source http://marcusjenkins.com)*



Arduino NANO *(source http://marcusjenkins.com)*



Arduino MEGA *(source http://marcusjenkins.com)*

1.2 Fonctions & bornes des cartes Arduino

SPI	MOSI	MISO	SCK	CS ou SS
UNO	11	12	13	10
NANO	11	12	13	10
MEGA	51	50	52	53

I2C	SDA	SCL
UNO	A4	A5
NANO	A4	A5
MEGA	20	21

PWM	PWM	
UNO	3, 5, 6, 9, 10 et 11	
NANO	3, 5, 6, 9, 10 et 11	
MEGA	2 à 13	44 à 46

Num*	A0	A1	A2	A3	A4	A5	A6	A7
UNO	14	15	16	17	18	19		
NANO	14	15	16	17	18	19	20	21

*Num * : Nom des broches analogiques utilisées en numérique (DIGITAL)*

1.3 Driver CH340 : connexion USB

Les cartes compatibles Arduino utilisent la puce CH340 pour fournir une connectivité USB. Il est nécessaire d'installer le pilote CH340 pour connecter une carte Arduino à un PC Windows : le convertisseur USB permet ainsi une communication via le port série de l'Arduino.

Une recherche sur un navigateur Internet permet de trouver des liens de téléchargement.

Par exemple :

- https://github.com/HobbyComponents/CH340-Drivers
- https://www.arduined.eu/ch340-windows-8-driver-download/
- http://283.mytrademe.info/ch340.html

1.4 Simulateurs en ligne

A. Il est possible de réaliser des montages Arduino basiques grâce à un simulateur en ligne gratuit (mais propriétaire). Ce simulateur, appelé « **123D Circuits** », est proposé par la société Autodesk : http://123d.circuits.io

B. Le site www.arduino.cc propose également un simulateur et programmateur en ligne. Il faut pour cela créer un compte à l'adresse : https://create.arduino.cc

1.5 Raccourcis clavier

Ctrl + S	→ Enregistre le projet
Ctrl + T	→ Met en forme du texte (formatage automatique)
Ctrl + R	→ Vérifie le code de programmation
Ctrl + U	→ Téléverse le code de programmation dans l'Arduino

1.6 Gestion du code et des commentaires

```
delay (2000) ; // Ajout du commentaire
```
→ Met en commentaire toute la suite de la ligne

```
/*
…
…
*/
```
→ Les symboles /*et */délimitent le code mis en commentaire

1.7 Principales fonctions Numériques Entrées / Sortie

pinMode(2, OUTPUT);	→ Configure la broche numérique N°2 en sortie
pinMode(2, INPUT);	→ Configure la broche numérique N°2 en entée
pinMode(2, INPUT_PULLUP);	→ Configure la broche numérique N°2 en entée avec l'activation de la résistance de rappel interne 'résistance de tirage)
digitalWrite(2, HIGH);	→ Met un niveau logique HIGH (HAUT en anglais) sur la broche numérique N°2
digitalWrite(2, LOW);	→ Met un niveau logique LOW (BAS en anglais) sur la broche numérique N°2
digitalRead(2);	→ Lit l'état de la broche numérique N°2 : valeur HIGHT (1) ou LOW (0)

1.8 Principales fonctions Analogiques Entrées / Sortie

analogRead (A2) ;	→ Lit l'état de la broche analogique N°A2 : valeur comprise entre 0 et 1023 pour une carte ayant une résolution de 10 bits Seules les broches A0 à Ax peuvent être lues en analogique
analogWrite (2, 255) ;	→ Envoie une onde PWN sur la broche N°2 : le cycle varie entre 0 (toujours à l'état BAS) et 255 (toujours à l'état HAUT) Toutes les broches ne sont pas compatibles avec le PWM La fréquence PWM d'Arduino est d'environ 500 Hz

1.9 Principales fonctions de Temps

delay (2000) ;	→ Met le programme en pause pendant la durée (en millisecondes) spécifiée en tant que paramètre
delayMicroseconds (2000) ;	→ Met le programme en pause pendant la durée (en microsecondes) spécifiée en tant que paramètre.
micros () ;	→ Renvoie le nombre de microsecondes depuis que la carte Arduino a commencé à exécuter le programme Ce nombre unsigned long débordera (reviendra à zéro) après environ 70 minutes
millis () ;	→ Renvoie le nombre de microsecondes depuis que la carte Arduino a commencé à exécuter le programme actuel Ce nombre unsigned long débordera (reviendra à zéro) après environ 50 jours

Astuce :
Pour éviter des bugs au démarrage de cartes Arduino compatibles, on peut ajout delay (2000) dès la première ligne de démarrage du setup() ;

Exemple :

```
unsigned long previousMillis = 0;
long interval = 1000 ;

void setup()
{
}

void loop()
{
    unsigned long currentMillis = millis();
    if (currentMillis – previousMillis >=  interval)
    {
        previousMillis = currentMillis;
        …
    }
}
```

1.10 Principales fonctions de nombre aléatoire

random (300) ;	→ Génère un nombre unsigned long pseudo-aléatoire entre 0 et 299
random (30 , 200) ;	→ Génère un nombre unsigned long pseudo-aléatoire entre 30 et 200
randomSeed (300) ;	→ Génère un nombre unsigned long pseudo-aléatoire en le faisant démarrer à un point arbitraire de sa séquence aléatoire

Exemple avec aucune connexion sur la broche A0 : **randomSeed(analogRead(A0));**

1.11 Principales fonctions de gestion des bits et octets

bit (n) ;	→ n: le bit dont la valeur à calculer
bitClear (x, n) ;	→ x: la variable numérique dont le bit est à effacer n: bit à effacer, à mettre à « 0 »
bitSet (x, n) ;	→ x: la variable numérique dont le bit est à écrire n: bit pour écrire la valeur « 1 »
bitWrite (x, n, b) ;	→ x: la variable numérique dont le bit est à lire n: bit à lire b: la valeur à écrire sur le bit (0 ou 1)
bitRead (x, n) ;	→ x: la variable numérique dont le bit est à lire n: bit à lire

1.12 Principales fonctions de gestion des interruptions externes

attachInterrupt(digitalPinToInterrupt(pin), Apel, mode) ;	→ Broche 2 ou 3 pour UNO, NANO Broche 2, 3, 18, 19, 20, 21 pour MEGA → Apel = fonction appelée lors de l'interruption → mode **LOW** pour déclencher l'interruption lorsque la broche est à l'état BAS mode **CHANGE** pour déclencher l'interruption lorsque la broche change de valeur **RISING** pour se déclencher lorsque la broche passe de l'état BAS à l'état HAUT **FALLING** pour se déclencher lorsque la broche passe de l'état HAUT à l'état BAS
noInterrupts() ;	→ Désactive les interruptions
interrupts() ;	→ Active les interruptions

Exemple :

```
byte ledPin = 13;
byte interruptPin = 2;
boolean state = LOW;

void setup()
{
        pinMode(ledPin, OUTPUT);
        pinMode(interruptPin, INPUT_PULLUP);
        attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

void loop()
{
        digitalWrite(ledPin, state);
}

void blink()
 {
        state = !state;
}
```

2 Les Variables et constantes

2.1 Variables

Les différents types de variables :

Type	Nombre	Valeurs	Nbre sur X bits	Nbre d'octets
int	Entier	-32 768 à +32 767	16 bits	2 octets
long	Entier	-2 147 483 648 à +2 147 483 648	32 bits	4 octets
char	Entier	-128 à +127	8 bits	1 octet
float	Décimale	$-3{,}4 \times 10^{38}$ à $+3{,}4 \times 10^{38}$	32 bits	4 octets
double	Décimale	$-3{,}4 \times 10^{38}$ à $+3{,}4 \times 10^{38}$	32 bits	4 octets
unsigned char	entier non négatif	0 à 255	8 bits	1 octet
unsigned int	entier non négatif	0 à 65 535	16 bits	2 octets
unsigned long	entier non négatif	0 à 4 294 967 295	32 bits	4 octets
byte	entier non négatif	0 à 255	8 bits	1 octet
word	entier non négatif	0 à 65 535	16 bits	2 octets
boolean	entier non négatif	0 ou 1	1 bit	1 octet
string	chaîne de caractère			

2.2 Gestion et conversion de variables

inString.**toInt() ;**	→ Convertit la chaîne de caractères *inString* en un nombre entier
inString.**length() ;**	→ Renvoie la longueur de la chaîne de caractères *inString*
char (mavariable **)**	→ Convertit la valeur *mavariable* dans le type de donnée char
byte (mavariable **)**	→ Convertit la valeur *mavariable* dans le type de donnée byte
int (mavariable **)**	→ Convertit la valeur *mavariable* dans le type de donnée int
long (mavariable **)**	→ Convertit la valeur *mavariable* dans le type de donnée long
float (mavariable **)**	→ Convertit la valeur *mavariable* dans le type de donnée float

2.3 Constantes

#define ledPin 3	→ Remplace tout texte *ledPin* avec la valeur *3* au moment de la compilation
const float pinLed = 3.2 **;**	→ Affecte à la constante *pinLed* la valeur « 3.2 »
const int pinLed = 3 **;**	→ Affecte à la constante *pinLed* la valeur « 3 »
const char pinLed = 3 **;**	→ Affecte à la constante *pinLed* la valeur « 3 »

3 Les Tableaux

3.1 Créer un tableau à 1 dimension

Il faut indiquer au moins le nombre de données ou les valeurs :

```
int maVariable [6] ;
int maVariable [] = {2, 4, 8, 3, 6} ;
int maVariable [6] = {2, 4, -8, 3, 2} ;
char message [6] = "Bonjour" ;
```

Pour accéder aux données du tableau définit ci-dessus :

maVariable[0] ;	→ Renvoie la valeur « 2 »
maVariable[1] ;	→ Renvoie la valeur « 4 »
maVariable[10] ;	→ n'est pas valide et contient une donnée aléatoire

3.2 Créer un tableau à 2 dimensions

Il faut indiquer au moins le nombre de colonnes ou les valeurs :

```
int maVariable [Ligne] [Colonne] ;

int maVariable [2] [5] ;

int maVariable [] =
                    {2, 4, 8, 3, 6} ,
                    {3, 5, 9, 4, 7} ;
```

3.3 Créer un tableau de chaînes de caractères

```
char *myStrings[] =
                     {
                    "Chaine de caractere 1",
                    "Chaine de caractere 2",
                    "Chaine de caractere 3",
                    "Chaine de caractere 4"
                    } ;
```

Appelle la chaine de caractère de rang « 2 » : **myStrings[1];**

3.4 *Déterminer la taille d'un tableau (nombre de lignes, colonnes)*

Exemple :

```
#define ROW_COUNT(array)    (sizeof(array) / sizeof(*array))
#define COLUMN_COUNT(array) (sizeof(array) / (sizeof(**array) * ROW_COUNT(array)))

int tableau[][] =  {{1, 2, 3, 4, 5, 6}, {9, 8, 7, 6, 5, 4}};

void setup()
{
int x = ROW_COUNT(tableau);
int y = COLUMN_COUNT(tableau);
}
```

4 Les Opérations

i++ ;	→ Incrémentation : i = i + 1 ;
i-- ;	→ Décrémentation : i = i - 1 ;
!x	→ Inversion : x = !x ;
x +=y ;	→ Correspond à x = x + y ;
x -= y ;	→ Correspond à x = x - y ;
X *= y ;	→ Correspond à x = x * y ;
X /= y ;	→ Correspond à x = x / y ;

variable = **map** (variable, 0, 1023, 0, 255);	→ Étalonne la *variable* entre 0 et 1023 sur une fourchette entre 0 et 255
variable = **constrain** (variable, 5, 20);	→ Renvoie la valeur *variable* si la *variable* a une valeur comprise entre 5 et 20 Renvoie la valeur 5 si la *variable est* inférieure à 5 Renvoie la valeur 20 si la *variable est* supérieure à 20
variable = **min** (variable, 20);	→ Renvoie la plus petite valeur entre *variable* et 20
variable = **max** (variable, 20);	→ Renvoie la plus grande valeur entre *variable* et 20
variable = **abs** (variable);	→ Calcul la valeur absolue de la donnée *variable*

5 Les Conditions

5.1 Opérations logiques

&&	→ … ET …
||	→ …OU …
!	→ NON

5.2 Comparaisons

==	→ Teste l'égalité entre deux variables
!=	→ Teste la différence entre deux variables
<	→ Teste l'infériorité d'une variable par rapport à une autre
>	→ Teste la supériorité d'une variable par rapport à une autre
<=	→ Teste l'infériorité ou l'égalité d'une variable par rapport à une autre
>=	→ Teste la supériorité ou l'égalité d'une variable par rapport à une autre

5.3 Conditions

`if ( macondition == 1 ) fonction1() ;`	→ Exécute l'instruction *fonction1()* si la variable *macondition* est égale à 1 (condition est VRAIE)
`if ( macondition == 1 )` `{` `  fonction1() ;` `}`	→ Exécute l'instruction *fonction1()* si la variable *macondition* est égale à 1 (condition est VRAIE)
`if ( macondition == 1 )` `{` `  fonction1() ;` `} else {` `  fonction2() ;` `}`	→ Exécute l'instruction *fonction1()* si la variable *macondition* est égale à 1 (condition est VRAIE) Sinon (condition est FAUX) exécute l'instruction *fontion2()*
`if ( macondition <10 )` `{` `  fonction1() ;` `} else if ( macondition > 20 )` `{` `  fonction2() ;` `} else {` `  fonction3() ;` `}`	→ Exécute l'instruction *fonction1()* si la variable *macondition* est strictement inférieure à 10 Sinon la variable *macondition* est strictement supérieure à 20, exécute l'instruction *fonction2()* sinon exécute l'instruction *fontion3()*
`while ( macondition <10 )` `{` `  fonction1() ;` `}`	→ Exécute l'instruction *fonction1()* tant que la variable *macondition* est strictement inférieure à 10

`while ( 1 )` `{` `  fonction1() ;` `}`	→ Exécute l'instruction *fonction1()* en infini : on sortira jamais de la boucle puisque "1" n'est pas une variable
`do` `{` `  fonction1() ;` `}` `while ( macondition <10 )`	→ Exécute l'instruction *fonction1()*au moins une fois tant que la variable *macondition* est strictement inférieure à 10 La boucle do / while fonctionne de la même façon que la boucle while, à la différence près que la condition est testée à la fin de la boucle
`switch ( mavariable )` `{` `case 1:` `  fonction1() ;` `  break;` `case 'A':` `  fonction2() ;` `  break;` `default:` `  fonction3() ;` `}`	→ Exécute l'instruction *fonction1()* si la variable *mavariable* est égale à 1 Exécute l'instruction *fonction2()* si la variable *mavariable* est égale au caractère A Sinon exécute l'instruction *fonction3()* *break;* est optionnel : permet de sortir du switch sans vérifier les conditions suivantes *default :* est optionnel : exécute l'instruction si les conditions ci-dessus n'ont pas été remplies
`for ( int i=0 ; i < 255 ; i++ )` `{` `  fonction1() ;` `}`	→ Exécute 255 fois l'instruction *fonction1()* ou tant que la variable « i » est strictement inférieur à 255 Dans une boucle For(), on doit définir l'initialisation, la condition pour sortir de la boucle et l'incrémentation

6 Port Série

6.1 Configuration

Dans la fonction SETUP, il faut d'abord ouvrir le port série et fixer le débit de communication.

```
void setup()
{
        Serial.begin(115200);
        …
}
```

La carte Méga dispose de plusieurs port série appelés Serial1, Serial2, Serial3.

Les cartes Arduino disposent d'une communication série matérielle sur les broches 0 et 1 (lesquelles sont également connectées à l'ordinateur via la connexion USB).

La librairie de communication série appelée « SoftwareSerial » a été développée pour permettre des communications série sur d'autres broches numériques de la carte Arduino, en utilisant un programme (cf. le code de la librairie) pour répliquer la même fonction de communication série.

6.2 Envoi de données

Pour envoyer des données sous forme de texte ASCII sur le port Série,

Serial.print("Texte");	→ Envoi de la chaine « Texte »
Serial.print(MaVariable**);**	→ Envoi de la valeur de la variable « MaVariable »
Serial.print(MaVariable **, DEC);**	→ Envoi de la valeur de la variable au format décimal
Serial.print(MaVariable **, BIN);**	→ Envoi de la valeur de la variable au format binaire
Serial.print(MaVariable **, HEX);**	→ Envoi de la valeur de la variable au format hexadécimal
Serial.print(MaVariable **, OCT);**	→ Envoi de la valeur de la variable au format octal
Serial.println();	→ println indique un retour chariot à la fin du texte envoyé. Équivalent à « **\n** »
Serial.print("**\t**");	→ Envoi d'une tabulation
Serial.print(3.14159265**, 4);**	→ Affiche 3.1415 Pour les nombres décimaux, le deuxième argument indique le nombre de chiffres après la virgule

6.3 Réception de données

Le programme Arduino dispose d'une fonction qui s'exécute de manière régulière avant chaque redémarrage de la fonction loop(). Il vous suffit d'ajouter une fonction nommée **serialEvent()** en dehors des fonctions setup() et du loop().

Exemple :

```
void setup()
{
        Serial.begin(115200);
        …
}

void loop()
{
        …
}

void serialEvent()              // déclaration de la fonction d'interruption sur la voie série
{
        while (Serial.available())
        {
        char inChar = (char)Serial.read();
        int c = (int)inChar ;
        …
        }
}
```

7 Débugger le programme

La compilation conditionnelle a pour but d'incorporer ou d'exclure des parties du code source dans le texte qui sera généré lors de la compilation.

La compilation conditionnelle permet d'adapter le programme au matériel ou à l'environnement sur lequel il s'exécute, ou d'introduire dans le programme des instructions de débogage.

```
#define Serial_DEBUG
```
→ Définition de la variable

```
#ifdef Serial_DEBUG
    …
#endif
```
Le code est compilé lors du téléversement du programme

```
// #define Serial_DEBUG
```
→ La définition de la constante est commentée

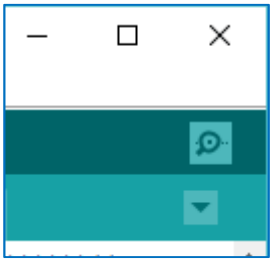
```
#ifdef Serial_DEBUG
    …
#endif
```
Le code n'est pas compilé lors du téléversement du programme

8 Inclure un fichier

Pour de gros projets, il devient utile de bien organiser le code. On peut séparer le code en différents fichiers afin d'avoir des entités logiques séparées les unes des autres.

8.1 Créer un fichier « .ino »

Pour créer un nouveau fichier « INO » dans l'IDE Arduino, il faut cliquer sur la petite flèche en haut de l'espace d'édition du code puis ensuite de cliquer sur "Nouvel Onglet" ou "New Tab" et renseigner le nom du fichier.



8.2 Créer un fichier « .h »

Pour créer un nouveau fichier « .h » dans l'IDE Arduino, il faut créer un fichier et l'enregistrer dans le répertoire du projet.

Il faut ensuite insérer « include » au début du programme avant la fonction setup().

```
#include "monfichier.h"
void setup()
{
        …
}
```

9 Les Fonctions

9.1 Fonction simple

Dans un programme, il est possible de séparer le programme en petits bouts afin d'améliorer la lisibilité et de faciliter le débogage. Une fonction est un « conteneur » mais différent des variables. Une variable ne peut contenir qu'un nombre, tandis qu'une fonction peut contenir un programme entier.

Une fonction est définie avec le code void et le nom de la fonction.

```
void setup()
{
}

void loop()
{
        nom_de_la_fonction() ;
}

void nom_de_la_fonction()
{
   // instructions
}
```

Pour appeler une fonction, il faut inscrire le nom de la fonction suivi de ses parenthèses « () » et ponctuer d'un point-virgule « ; » dans le bloc « { } » de la fonction setup() ou loop().

9.2 Fonction typée

Une fonction void ne peut pas renvoyer un résultat.

Une fonction typée peut renvoyer un résultat.

Exemple :

```
int calcul = 0;

void loop()
{
	calcul = 10 * maFonction();
}

int maFonction()
{
	int resultat = 44 ;
	return resultat;
}
```

La fonction peut renvoyer une variable de type int, booléen ou float.

9.3 Fonction avec paramètres

On peut définir la fonction avec son nom, le type de la valeur retournée et le type des différents paramètres.

Exemple :

```
int maFonction (int param1, int param2, int param3)
{
	int resultat = 0;
	switch(param3)
	{
		case 0 :
		resultat = param1 + param2;
		break;

		case 1 :
		resultat = param1 - param2;
		break;

		default :
		resultat = 0;
	}
	return resultat;
}
```

10 Économiser de la mémoire SRAM

10.1 Serial.print

Le type String permet de stocker une chaine de caractère.

Les chaines de caractères prennent sont consommateurs de SRAM.

Ces chaines de caractères utilisent de la place dans la mémoire Flash (l'image du programme) et elles sont ensuite copiées en SRAM lors de l'initialisation des variables statiques.

Il a été développé la macro F() qui offre une solution simple qui indique au compilateur de garder la chaine de caractère dans la PROGMEM (la mémoire flash qui stocke le programme).

Il faut alors enfermer la chaine littérale dans la macro F().

Exemple :

```
Serial.println(F("mon texte"));

lcd.println(F("mon texte"));
```

10.2 Progmem

On peut réduire l'empreinte mémoire d'un programme Arduino grâce à l'extension PROGMEM.
Il faut d'abord appeler la bibliothèque prévue à cet effet et définir les constantes.

Exemple :

```
#include <avr/pgmspace.h>

const char* PROGMEM ERROR_MSG_NO_MORE_COFFEE = "Plus de café dans la
cafetière";
const char* PROGMEM ERROR_MSG_PEBCAK = "Erreur d'interface chaise-clavier";
const char* PROGMEM ERROR_MSG_EAT_BY_DOG = "Le chien a mangé mon devoir";

const char* const PROGMEM ERROR_MESSAGES[] =
        {
        ERROR_MSG_NO_MORE_COFFEE,
        ERROR_MSG_PEBCAK,
        ERROR_MSG_EAT_BY_DOG
        };
```

Pour lire une constante :

```
const byte PROGMEM tableau[] =
{
// ...
};
```

→ Définition de la constante

```
byte valeur = pgm_read_word(tableau + index);
```

Lit un octet

```
const int PROGMEM tableau[] =
{
// ...
};
```

→ Définition de la constante

```
int valeur = pgm_read_word(tableau + index);
```

Lit un entier sur 16 bits

```
const long PROGMEM tableau[] =
{
// ...
};
```

→ Définition de la constante

```
long valeur = pgm_read_word(tableau + index);
```

Lit un entier sur 32 bits

```
const float PROGMEM tableau[] =
{
// ...
};
```

→ Définition de la constante

```
float valeur = pgm_read_word(tableau + index);
```

Lit un nombre à virgule

```
const char* PROGMEM message = "Mon message";
const char* const PROGMEM tableau[] = {
message,
};
```

→ Définition de la constante
Définition du tableau

```
char valeur[32];
strcpy_P(valeur, (char*) pgm_read_word(&(tableau[index])));
```

Définition de la variable
Lit la chaîne de caractères

11 Gestion de la mémoire EEPROM

Pour utiliser la mémoire Eeprom, il faut appeler la bibliothèque :

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

11.1 Écrire / Modifier

`EEPROM.write(adresse, valeur) ;` `EEPROM.write(0, 5) ;`	→ Écrit la valeur « 5 » à l'adresse 0 de la mémoire
`EEPROM.update(adresse, valeur) ;` `EEPROM.update(0, 5) ;`	→ Si besoin, modifie la valeur « 5 » à l'adresse 0 de la mémoire
`EEPROM.put(adresse, variable) ;` `EEPROM.put(0, 12.51) ;`	→ Écrit la valeur « 12.51» à partir de l'adresse 0 de la mémoire

11.2 Lire

`EEPROM.read(adresse) ;` `EEPROM.read(0) ;`	→ Écrit la valeur à l'adresse 0 de la mémoire
`float variable;` `EEPROM.get(adresse, variable) ;`	
`float variable;` `EEPROM.get(0, variable) ;`	→ Lit la valeur de la variable de type float à partir de l'adresse « 0 »

La fonction « `sizeof( mavariable )` » permet de connaitre la taille en octet d'une variable

12 Quelques bibliothèques utiles…

`#include <Servo.h>`	→ Commande de servomoteurs
`#include <Bounce2.h>`	→ Gestion des rebonds pour les boutons poussoirs
`#include <SoftwareSerial.h>`	→ Ajout de ports Série sur une carte UNO / NANO par exemple
`#include <Keypad.h>`	→ Gestion d'un clavier ou d'une matrice de boutons poussoirs
`#include <lcd.h>`	→ Gestion d'un écran LCD
`#include <LiquidCrystal_I2C.h>`	→ Gestion d'un écran LCD avec I2C
`#include <EEPROM.h>`	→ Gestion de la mémoire Eeprom
`#include <MemoryFree.h>`	→ Indique la mémoire disponible au moment de l'exécution

…

13 Pour aller plus loin…

13.1 Installer le logiciel XLoader

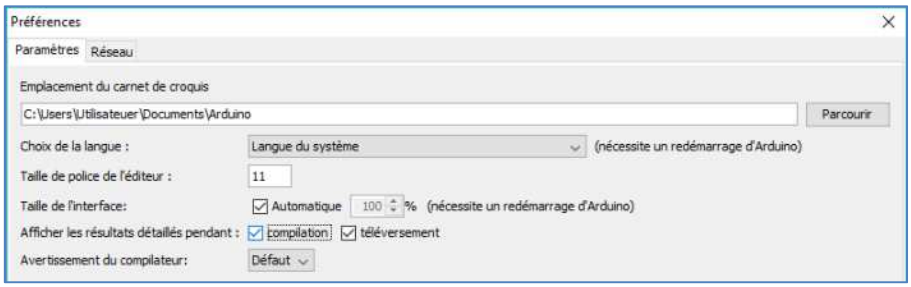
XLoader est un petit programme qui permet de télécharger les propres fichiers « * .hex » sur des cartes Arduino. Le programme s'exécute en version portable. Voici le lien de téléchargement : http://russemotto.com/xloader/

13.2 Créer un fichier « HEX »

Lors de la compilation d'un programme « INO », il est créé un répertoire temporaire contenant toutes les librairies utilisées. Il est également généré un fichier « HEX » qui pourra être téléchargé dans une autre carte Arduino à l'aide de l'application « XLoader » sans avoir besoin d'installer les librairies, code sources, …

Pour visualiser le chemin du répertoire où sont générés les fichiers :

1. Ouvrir le programme ARDUINO
2. Dans le menu « Fichier », cliquer sur « Préférences » ou « Ctrl + Virgule »
3. Cocher au moins une option « Compilation » ou « Téléversement » et fermer la fenêtre en cliquant sur « Ok »



4. Compiler le programme
5. Dans la fenêtre de sortie en pied de page, repérer la ligne :
   C:\Users\VOTRENOM**AppData\Local\Temp**arduino_build_XXXXXX/monprojet**.ino.hex**
6. Noter le chemin et ouvrir le répertoire dans l'explorateur de fichiers

13.3 Télécharger ou Uploader un fichier « HEX » dans une carte Arduino

1. Connecter à l'ordinateur la carte ARDUINO UNO, NANO ou MEGA à programmer
2. Lancer l'application « XLoader »
3. Sélectionner le port COM de l'Arduino dans le menu déroulant en bas à gauche
4. Sélectionner la carte ARDUINO dans la liste déroulante « Device »
5. Vérifier que l'application XLoader a défini le débit en bauds comme suit :
   i. *57600 pour Duemilanove / Nano (ATmega 328)*
   ii. *115200 pour Uno (ATmega 328)*
   iii. *115200 pour Mega (ATMEGA2560)*
6. Renseigner le nom et l'adresse du fichier « HEX » dans le champ « Hex File »
7. Cliquer sur le bouton « Upload » pour charger le fichier. Les LED RX/TX de la carte Arduino doivent clignoter. Après un certain temps, le programme est téléversé.