Introduction aux Systèmes embarqués

CHAP I : Présentation de la plateforme ARDUINO

1. Présentation de la plateforme ARDUINO :

1.1. Introduction :

Arduino est un projet qui a été initié par un groupe d'enseignants et d'étudiants d'une école italienne en 2004–2005 dans le but de concevoir et fabriquer des solutions matérielle et logicielles à faible coût.

Arduino est aussi le nom des cartes électroniques conçues pour un accès simple et peu couteux à l'informatique embarquée. De plus, elles sont entièrement libres de droit, autant sur l'aspect du code source (Open Source) que sur l'aspect matériel (Open Hardware). Ainsi, il est possible de refaire une carte Arduino personnalisée dans le but de l'améliorer ou d'enlever des fonctionnalités inutiles au projet.

Le logiciel de développement IDE (Environnement de Développement Intégré) qui porte le même nom est un logiciel multi OS qui permet l'édition, la compilation et le transfert du programme dans la carte électronique via le port USB.

Les cartes Arduino sont basées sur des microcontrôleurs. Un microcontrôleur est une petite unité de calcul accompagné de mémoire, de ports d'entrée/sortie et de périphériques permettant d'interagir avec son environnement. Parmi les périphériques, on recense généralement des Timers, des convertisseurs analogique-numérique, des liaisons Séries...

Voici une liste non exhaustive des applications possible réalisées grâce à Arduino :

- Contrôler des appareils domestiques.
- Donner une "intelligence" à un robot.
- Réaliser des jeux de lumières.
- Permettre à un ordinateur de communiquer avec une carte électronique et différents capteurs.
- Télécommander un appareil mobile

1.2. Types de cartes :

Il y a trois types de cartes :

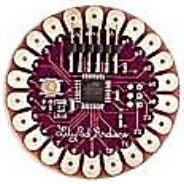
- Les dites « officielles » qui sont fabriquées en Italie par le fabricant officiel : Smart Projects.
- Les dites « compatibles » qui ne sont pas fabriquées par Smart Projects, mais qui sont totalement compatibles avec les Arduino officielles.
- Les autres fabriquées par diverse entreprise et commercialisées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

Selon les caractéristiques techniques, on trouve plusieurs types de cartes ARDUINO (voir figure suivante) :


Arduino Uno


Arduino Leonardo


Arduino Mega 2560


Arduino LilyPad


Arduino Mega ADK

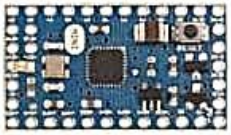

Arduino Fio


Arduino Ethernet

Arduino Pro


Arduino BT


Arduino Nano


Arduino Mini

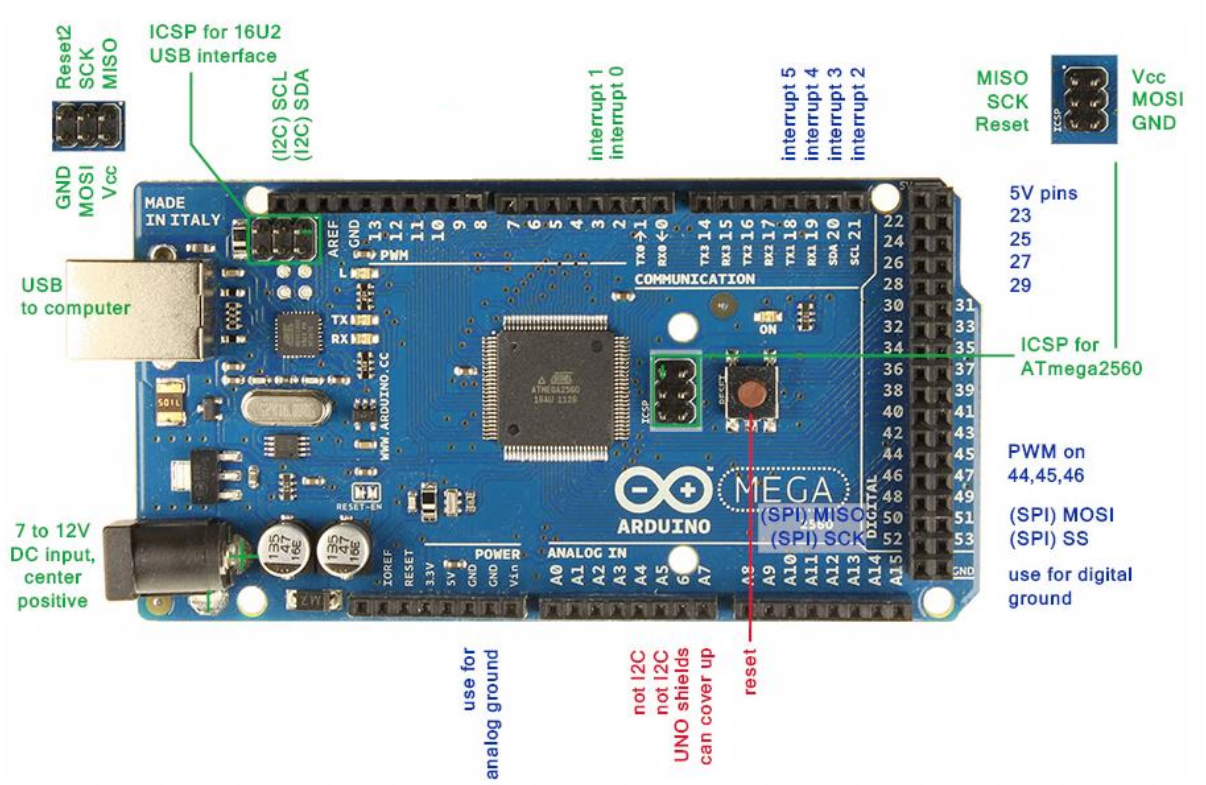

Arduino Pro Mini

1.3. Caractéristiques techniques de la carte Arduino MEGA :

Pendant toutes les parties de ce cours/TP on sera amené à utiliser la carte Arduino Mega 2560 qui est basée sur un microcontrôleur ATMega1280 cadencé à 16 MHz. Elle dispose de 54 E/S dont 14 PWM, 16 analogiques et 4 liaisons séries. Elle est idéale pour des applications exigeant des caractéristiques plus complètes que la Uno. Des connecteurs situés sur les bords extérieurs du circuit imprimé permettent d'enficher une série de modules complémentaires. Le tableau suivant regroupe les caractéristiques techniques de cette carte et la figure de la page suivante représente son schéma :

Tension de fonctionnement	5 V
Tension d'alimentation (recommandée)	7- 12 V
Tension d'alimentation (limites)	6 - 20V
Nombre d'E/S	54 (dont 14 pouvant générer des signaux PWM)
Nbr ports "Analogique/Numérique"	16
Courant max. par E/S	40 mA
Courant pour broches 3.3 V	50 mA
Mémoire Flash	256 KB dont 8 KB utilisé par le boot-loader
SRAM	8 KB
EEPROM	4 KB
Vitesse horloge	16 MHz
Dimensions	101,52 x 53,3 mm
Poids	37 g

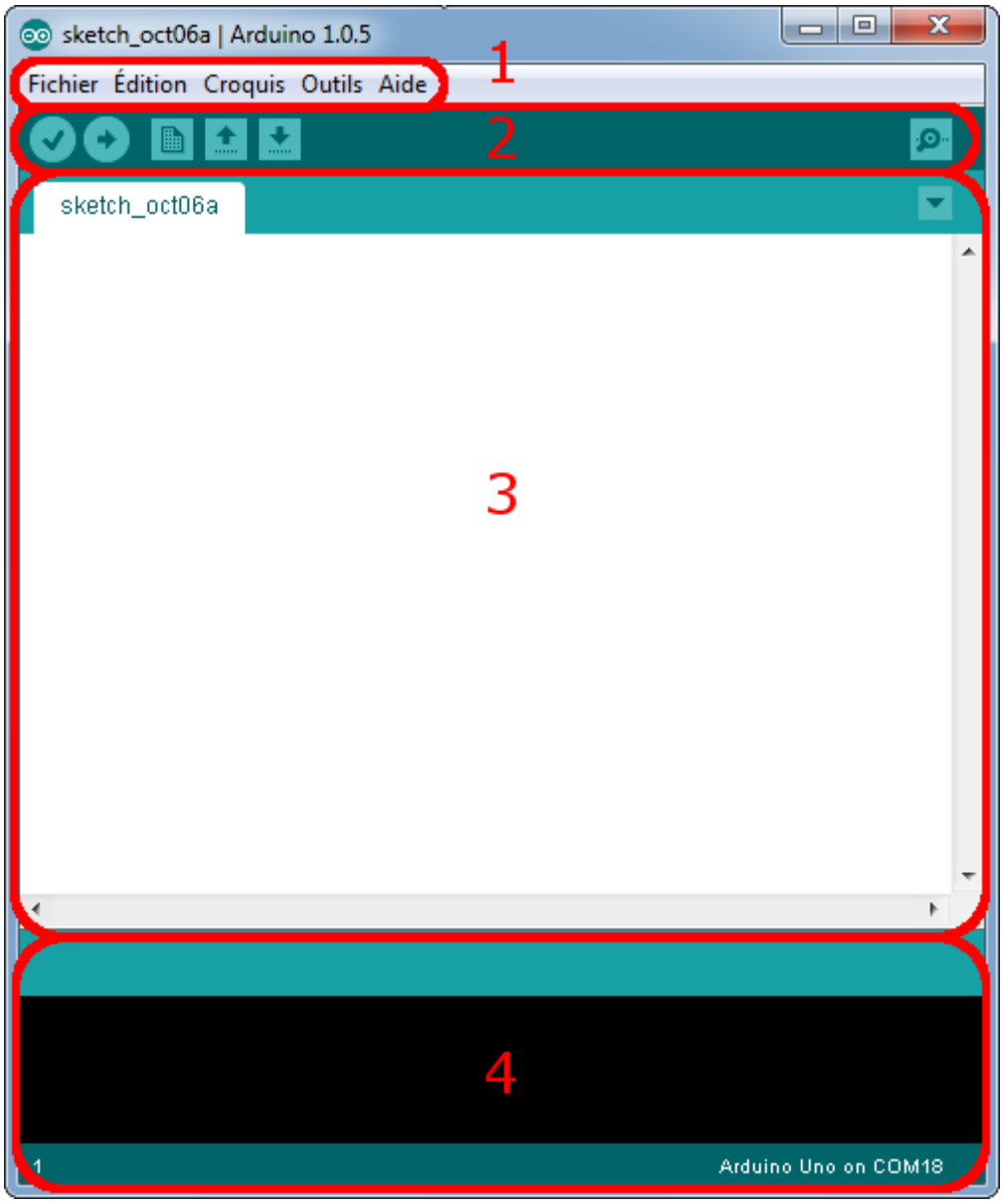
Elle peut se programmer avec le logiciel Arduino qui est téléchargeable gratuitement. Le contrôleur ATMega1280 contient un boot-loader qui permet de modifier le programme sans passer par un programmateur.



1.4. Présentation du logiciel Arduino IDE :

Le logiciel Arduino IDE est libre et gratuit. Il est distribué sur le site d'Arduino (compatible Windows, Linux et Mac). D'autres alternatives existent pour développer pour Arduino (extensions pour CodeBlocks, Visual Studio, Eclipse, XCode, etc.) mais nous n'aborderons dans ce cours que l'IDE officiel.

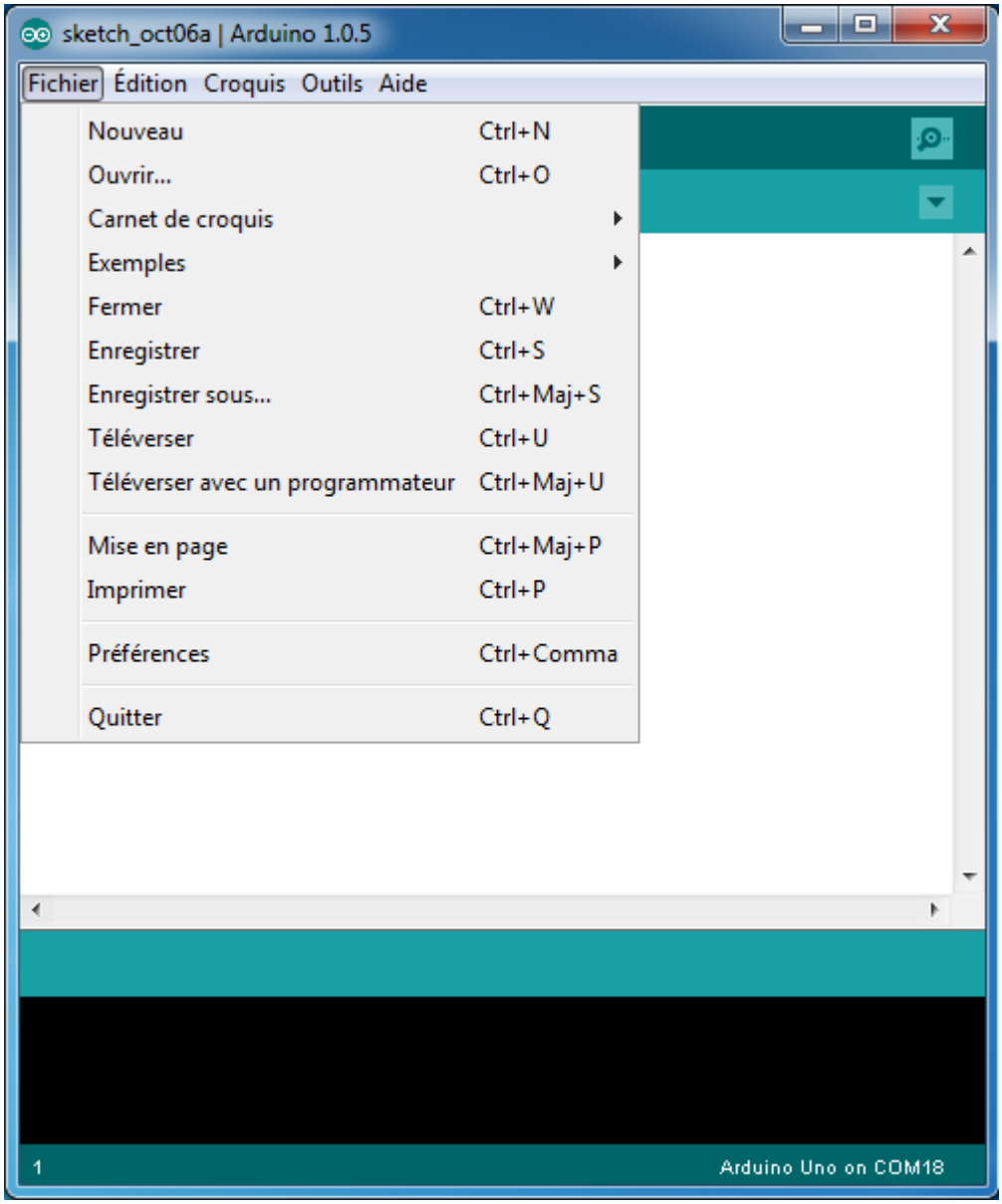
L'interface de l'IDE Arduino est simple (voir figure suivante), il offre une interface minimale et épurée pour développer un programme sur les cartes Arduino. Il est doté d'un éditeur de code avec coloration syntaxique [3] et d'une barre d'outils rapide [2]. Ce sont les deux éléments les plus importants de l'interface, c'est ceux que l'on utilise le plus souvent. On retrouve aussi une barre de menus [1] plus classique qui est utilisée pour accéder aux fonctions avancées de l'IDE. Enfin, une console [4] affichant les résultats de la compilation du code source, des opérations sur la carte, etc.



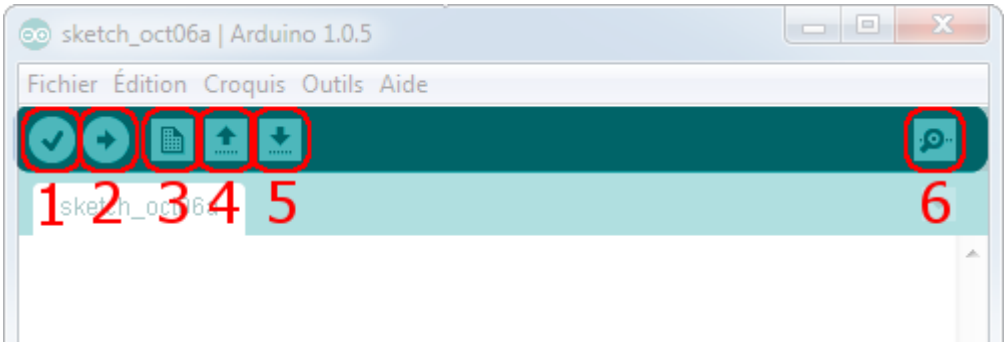
Le menu le plus utilisé dans la programmation et la configuration des cartes est le menu « fichier » (voir figure suivante) :

- Carnet de croquis : Ce menu regroupe les fichiers déjà réalisés.
- Téléverser : Permet d'envoyer le programme sur la carte Arduino.
- Téléverser avec un programmateur : Idem que si dessus, mais avec l'utilisation d'un programmateur (vous n'en aurez que très rarement besoin).
- Préférences : Vous pourrez régler ici quelques paramètres du logiciel.

Le reste des menus n'est pas intéressant pour l'instant et il sera détaillé pendant les travaux pratiques.



Il existe aussi des boutons d'accès rapide aux fonctions usuelles (voir figure suivante) :



- Bouton 1 : Ce bouton permet de vérifier le programme.
- Bouton 2 : Charge le programme dans la carte Arduino.
- Bouton 3 : Crée un nouveau fichier.

- Bouton 4 : Ouvre un fichier.
- Bouton 5 : Enregistre le fichier.
- Bouton 6 : Ouvre le moniteur série.

2. Langage Arduino :

Le langage Arduino est inspiré de plusieurs langages. On retrouve notamment des similarités avec le C, le C++ et le Java. Le langage impose une structure particulière typique de l'informatique embarquée. La syntaxe de ce langage de programmation désigne l'ensemble des règles d'écriture, on présente ci-après les fonctionnalités les plus utilisées et celles dont on aura besoins pendant ces travaux pratiques.

2.1. 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 grandes parties.

```
//fonction d'initialisation de la carte
void setup()
{
        //contenu de l'initialisation
}
//fonction principale, elle se répète (s'exécute) à l'infini
void loop()
{
        //contenu de votre programme
}
```

Dans ce code se trouvent deux fonctions :

La fonction ***setup :*** contiendra toutes les opérations nécessaires à la configuration de la carte (directions des entrées sorties, débits de communications série, etc.).

La fonction ***loop*** : elle est exécutée en boucle après l'exécution de la fonction *setup*. Elle continuera de boucler tant que la carte n'est pas mise hors tension ou redémarrée (par le bouton reset). Cette boucle est absolument nécessaire sur les microcontrôleurs étant donné qu'ils n'ont pas de système d'exploitation.

En effet, si l'on omettait cette boucle, à la fin du code produit, il sera impossible de reprendre la main sur la carte Arduino qui exécuterait alors du code aléatoire. Au niveau de la syntaxe, on retrouve des similarités avec les langages précédemment cités.

2.2. Les variables :

Une variable est une façon de nommer et de stocker une valeur pour une utilisation ultérieure par le programme, tel que les données issues d'un capteur ou une valeur intermédiaire utilisée dans un calcul.

Avant d'être utilisées, toutes les variables doivent être déclarées. Déclarer une variable signifie :

- définir son type, c'est à dire le genre de données qu'elle contiendra (nombre entiers, à virgule, ou caractère) et sa taille.
- la nommer (lui donner un nom).
- lui donner éventuellement une valeur initiale (initialiser la variable). A noter que les variables peuvent ne pas être initialisées (avoir une valeur attribuée) lors de leur déclaration, mais cela n'est pas souvent utile.

La déclaration des variables se fait généralement dans **l'espace global** (avant la fonction setup) (de façon à partager les variables les plus importantes entre les deux fonctions principales).

Concernant les types des variables qu'on peut utiliser, on retrouve les types de base suivants :

Nom	Contenu	Taille (en octet)	Plage de valeurs
(unsigned) char	Entier ou caractère	1	(0->255) -128 -> 127
(unsigned) int	Entier	2	(0->65 535) -32 768 -> 32 767
(unsigned) long	Entier	4	(0 -> 4 294 967 295) -2 147 483 648 -> 2 147 483 647
float/double	Nombre à virgule flottante	4	-3,4028235E+38 -> 3,4028235E+38
String	Chaine de caractères (Objet)	variable	Aucune
boolean	Booléen	1	True / False

- Exemple de déclarations :

```
const int constante = 12 ;
float univers = 42.0;
char lettre = 'b' ;
String chaine = "Hello World " ;
long tableau[12] ;
boolean vrai = true ;
```

2.3. Les opérations de base :

2.3.1. L'addition :

Le code suivant permet de faire l'addition entre deux nombres et stocker le résultat dans la variable x :

```
//définition de la variable x
int x = 0;
//on change la valeur de x par une opération simple
x = 12 + 3;
// x vaut maintenant 12 + 3 = 15
```

On peut faire aussi l'addition entre variables :

```
//définition de la variable x et assignation à la valeur 38
int x = 38;
int y = 10;
int z = 0;
//faisons une addition
z = x + y;
// on a donc z = 38 + 10 = 48
```

2.3.2. La soustraction :

On peut reprendre les exemples précédents, en faisant une soustraction :

```
/définition de la variable x
Int x = 0;
//on change la valeur de x par une opération simple
x = 12 - 3;
// x vaut maintenant 12-3 = 9
```

Soustraction de variables :

```
int x = 38;//définition de la variable x et assignation à la valeur 38
int y = 10;
int z = 0;
z= x-y;
// on a donc z = 38-10= 28
```

2.3.3. La multiplication :

```
int x = 0;
int y = 10;
int z = 0;
x = 12 * 3;// x vaut maintenant 12 * 3 = 36
z = x * y;// on a donc z = 36 * 10 = 360
// on peut aussi multiplier (ou toute autre opération) un nombre et une
variable :
z = z * ( 1/ 10 );//soit z = 360 * 0.1 = 36
```

2.3.4. La division :

```
float x = 0;
float y = 15;
float z = 0;
x = 12 / 2;// x vaut maintenant 12 / 2 = 6
z= y / x;// on a donc z = 15 / 6 = 2.5
```

Attention cependant, si vous essayer de stocker le résultat d'une division dans une variable de type char, int ou long, le résultat sera stocké sous la forme d'un entier arrondi au nombre inférieur.

```
float x = 0;
float y = 15;
intz = 0;
x = 12 / 2;// x vaut maintenant 12 / 2 = 6
z= y / x;// on a donc z = 15 / 6 = 2 !
```

2.3.5.Le modulo :

Cette opération permet d'obtenir le reste d'une division :

```
18 % 6 // le reste de l'opération est 0,car il y a 3*6 ds 18 dc 18-18= 0
18 % 5 // le reste de l'opération est 3,car il y a 3*5 ds 18 dc 18-15= 3
```

```
int x = 24;
int y = 6;
int z = 0;
z = x % y; // on a donc z = 24 % 6 = 0 (car 6 * 4 = 24)
```

2.3.6.L'incrémentation :

```
var = 0;
var++; //c'est cette ligne de code qui nous intéresse
```

"var++;" Revient à écrire : "var = var + 1;" En fait, on ajoute le chiffre 1 à la valeur de *var*. Et si on répète le code un certain nombre de fois, par exemple 30, et bien on aura *var* = 30.

2.3.7.La décrémentation :

```
var = 30;
var--;
//décrémentation de var
```

2.4. Les opérations composées :

Ces opérations permettent de faire des raccourcis lorsqu'on veut effectuer une opération sur une même variable :

```
int x, y;
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;
```

```
int var = 10;
//opération 1
var = var + 6;
var += 6;//var = 16
//opération 2
var = var - 6;
var -= 6;//var = 4
//opération 3
var = var * 6;
var *= 6;//var = 60
//opération 4
var = var / 5;
var/= 5;//var = 2
```

2.5. Les conditions :

Les conditions permettent de tester des variables et exécuter des instructions si le test est vérifié. On appelle structure conditionnelle les instructions qui permettent de tester si une condition est vraie ou non. Pour tester des variables, il faut connaître quelques symboles, voir tableau suivant :

Symbole	A quoi il sert	Signification
==	Ce symbole, composé de deux égales, permet de tester l'égalité entre deux variables	... est égale à ...
<	Celui-ci teste l'infériorité d'une variable par rapport à une autre	...est inférieur à...
>	Là c'est la supériorité d'une variable par rapport à une autre	...est supérieur à...
<=	teste l'infériorité ou l'égalité d'une variable par rapport à une autre	...est inférieur ou égale à...
>=	teste la supériorité ou l'égalité d'une variable par rapport à une autre	...est supérieur ou égal à...
!=	teste la différence entre deux variables	...est différent de...

2.5.1. If

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

```
if (uneVariable> 50)
{
  // faire quelque chose
}
```

Dans cet exemple, le programme va tester si la variable *une Variable* est supérieure à 50. Si c'est le cas, le programme va réaliser une action particulière. Autrement dit, si l'état du test entre les parenthèses est vrai, les instructions comprises entre les accolades sont exécutées. Sinon, le programme se poursuit sans exécuter ces instructions.

Les accolades peuvent être omises après une instruction if. Dans ce cas, la suite de la ligne (qui se termine par un point-virgule) devient la seule instruction de la condition. Tous les exemples suivants sont corrects :

```
if (x > 120)  digitalWrite(LEDpin, HIGH);
if (x > 120)
digitalWrite(LEDpin, HIGH);
if (x > 120){digitalWrite(LEDpin, HIGH); }// ttes ces formes sont correctes
```

2.5.2. if /else :

L'instruction if/else (si/sinon en français) permet un meilleur contrôle du déroulement du programme, en permettant de grouper plusieurs tests ensemble. Par exemple, une entrée analogique peut-être testée et une action réalisée si l'entrée est inférieure à 500, et une autre action réalisée si l'entrée est supérieure ou égale à 500. Le code ressemblera au programme suivant :

```
if (brocheCinqEntree< 500)
{
  // action A
}
else
{
  // action B
}
```

Noter qu'un bloc else if peut être utilisé avec ou sans bloc de conclusion else et vice versa. Un nombre illimité de branches else if est autorisé.

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

2.5.3. Switch/case :

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

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

```
switch (var) {
        case 500:
      //si la variable vaut 500, effectuer une action
        break;
        case 900:
      //si la variable vaut 900, effectuer une action
        break;
        default:
      //si aucune condition n'est vraie,  alors on exécute cette partie
      // il n'est pas obligatoire d'avoir la partie "default:"
  }
```

Paramètres :

- var : variable dont vous voulez tester l'état.
- default : si aucune autre condition vraie n'est rencontrée, le code default sera exécuté.
- break : sans l'instruction break, l'instruction switch passera en revue toutes les instructions.
- case : à la recherche d'une condition vraie. Si une est trouvée, le code sera exécuté. L'instruction break indique à l'instruction switch d'arrêter de rechercher des conditions vraies, et fait sortir de la structure de choix.

Il existe des opérateurs qui permettent de tester une combinaison de conditions (plusieurs conditions à la fois pour réaliser une action).

- && qui signifie ET
- || qui signifie OU
- ! qui signifie NON

2.6. Les boucles

2.6.1.La boucle for :

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

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

```
for (initialisation; condition; incrémentation) {

//instruction(s)à exécuter ;
}
```

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

Exemple :

```
// Eteint progressivement une LED en utilisant une broche PWM (impulsion)
int PWMpin = 10; // LED en série avec une résistance de 1k sur la broche 10
void setup()
{
  // aucune initialisation nécessaire
}
void loop()
{
   // boucle incrémentant la variable i de 0 à 255, de 1 en 1
   for (int i=0; i <= 255; i++
   {
      // impulsion de ratio HAUT/BAS fonction de i sur la broche 10
        analogWrite(PWMpin, i);
        delay(10); // pause de 10ms
   } // fin de la boucle for
}
```

2.6.2.La boucle while :

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

```
while(expression){ // tant que l'expression est vraie
 // instructions à effectuer
}
```

2.7. Les fonctions :

Dans un programme, les lignes sont souvent très nombreuses. Il devient alors impératif de séparer le programme en petits bouts afin d'améliorer la lisibilité de celui-ci, en plus d'améliorer le fonctionnement et de faciliter le débogage.

Une fonction est un "morceau" de programme délimité :

- qui peut ou non recevoir une ou plusieurs valeurs utilisées pendant son exécution : ce(s) valeur(s) sont appelées argument(s),
- et qui peut ou non renvoyer une valeur calculée pendant son exécution : ceci est défini par ce que l'on appelle le type de la fonction.

D'une manière générale, en langage Arduino (tout comme en langage C), toute fonction s'écrit sous la forme :

```
Type nom_fonction (arguments) {

  // ici le code de la fonction
}
```

Le code de la fonction est compris entre des accolades qui sont en quelque sorte les "bornes" délimitant la fonction.

Dans sa forme la plus simple, une fonction n'utilise aucune valeur (càd n'utilise aucun argument) et ne renvoie aucune valeur (on dira qu'elle est de type void). Les fonctions loop() et setup() sont précisément des fonctions simples qui toutes les deux :

- sont de type void (ceci veut dire qu'elles ne renvoient aucune valeur).
- ne reçoivent aucun argument : on ne mettra rien entre les parenthèses.

Exemple :

On souhaite écrire une fonctionne qui retourne la somme de deux nombres entiers. Cette fonction donc sera de type int et aura besoin de deux paramètres de même type :

```
int maFonction(intparam1, intparam2)
```

```
int x = 64;
int y = 192;
void loop()
{
        maFonction(x, y);
}
int maFonction(int param1, int param2)
{
        int somme = 0;
        somme= param1 + param2;
//somme = 64 + 192 = 255
        return somme;
}
```

Si on souhaite faire plusieurs opérations avec une seule fonction on peut utiliser le code suivant :

```
unsigned char operation= 0;
int x = 5;
int y = 10;
void loop()
{
//le paramètre "opération" donne le type d'opération à faire
        maFonction(x, y, operation);
}
int maFonction(int param1, int param2, int param3)
{
        int resultat= 0;
        switch(param3)
        {
        Case 0 : //addition, resultat = 15
        resultat= param1 + param2;
        break;
        case 1 : //soustraction, resultat = 5
        resultat= param1-param2;
        break;
        case 2 : //multiplication, resultat = 50
        resultat= param1 * param2;
        break;
        case 3 : //division, resultat = 0 (car nombre entier)
        resultat= param1 / param2;
        break;
```

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

2.8. Les tableaux :

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

Un tableau contient des éléments de même type. On le déclare donc avec un type semblable, et une taille représentant le nombre d'éléments qu'il contiendra. Par exemple, si on souhaite déclarer un tableau pour stocker les notes de 20 étudiants d'une classe, on utilisera la méthode suivante :

```
float notes[20];
```

Dans d'autres cas, on peut utiliser l'une des méthodes suivantes :

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[5] = {2, 4, -8, 3, 2};
char message[6] = "hello";
```

On peut aussi déclarer un tableau de variables sans les initialiser, tel que le tableau myInts dans l'exemple précèdent.

Dans le tableau myPins de l'exemple, nous déclarons un tableau sans déclarer explicitement une taille. Le compilateur compte les éléments et crée un tableau de la taille appropriée.

Enfin, on peut à la fois initialiser et dimensionner un tableau, comme avec le tableau mySensVals de l'exemple. Noter que lorsque l'on déclare un tableau de type char, un élément supplémentaire doit être prévu lors de l'initialisation pour y stocker le caractère nul de fin de chaîne.

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

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

```
int myArray[10]={9,3,2,4,3,2,7,8,9,11};
     // myArray[9]    contient 11
     // myArray[10]   n'est pas valide et contient une information
aléatoire (autre adresse mémoire)
```

Les tableaux sont souvent utilisés à l'intérieur de boucle for, où le compteur de la boucle est utilisé en tant qu'index pour chaque élément du tableau. Par exemple, pour afficher les éléments d'un tableau sur le port série, on peut écrire quelque chose comme le programme suivant :

```
int i;
  for (i = 0; i < 5; i = i + 1) {
  Serial.println(myPins[i]);
}
```

Le code suivant est celui d'un exemple de calcul de la moyenne et de la meilleure note dans une classe de 20 étudiants :

```
float meilleurNote(float tableau[], int nombreEleve)
{
        int i = 0;
        int max = 0;//variable contenant la future meilleure note
        for(i=0;i<nombreEleve, i++)
        {
        if(tableau[i] >max) //si la note lue est meilleure que la meilleure
actuelle
        {
        max = tableau[i];//alors on l'enregistre
        }
        }
        return max;//on retourne la meilleure note
}

float moyenneNote(float tableau[], int nombreEleve)
{
        int i = 0;
        double total = 0;//addition de toutes les notes
        float moyenne = 0;//moyenne des notes
        for(i = 0 ;i<nombreEleve ;i++)
        {
        total = total + tableau[i];
        }
        moyenne= total / nombreEleve;
        return moyenne;
}
```

On peut aussi trouver un tableau à deux dimensions (remarquer la syntaxe) :

```
char touches[LIGNES][COLONNES] = { // tableau de char à 2 dimensions
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
```