



## STI2D – Enseignement de spécialité SIN

# DOCUMENT RESSOURCE : CARTE DE DEVELOPPEMENT ARDUINO





# SOMMAIRE

<b>1 – PRESENTATION .....</b>	<b>3</b>
<b>2 – LA CARTE ARDUINO UNO. ....</b>	<b>4</b>
2.1 – Présentation .....	4
2.2 – Différents éléments de la carte Arduino Uno .....	4
2.3 – Principales caractéristiques .....	5
2.4 – Alimentation .....	5
2.5 – Les entrées/sorties .....	5
2.6 – Les mémoires.....	6
2.7 – Communication .....	7
2.8 – Reset .....	7
2.9 – Protection de surintensité USB .....	7
2.10 – Schéma structurel de la carte « Arduino Uno » .....	8
<b>3 – PROGRAMMATION DE LA CARTE ARDUINO .....</b>	<b>9</b>
3.1 – Interface de programmation .....	9
3.2 – Structure d'un programme.....	10
3.3 – Données, variables et constantes.....	11
3.4 – Les opérateurs .....	12
3.5 – Fonctions mathématiques.....	13
Fonctions « min », « max », « constrain », « abs », « sqrt » .....	13
Fonction « map ».....	14
3.6 – Fonctions de gestion du temps .....	14
Fonctions « delay », « delayMicroseconds » .....	14
3.7 – Fonctions de gestion des E/S numériques .....	14
Fonction « pinMode ».....	14
Fonctions « digitalWrite », « digitalRead » .....	15
3.8 – Fonction de gestion des sorties analogiques .....	15
Fonction « analogWrite ».....	15
3.9 – Fonctions de gestion des entrées analogiques .....	15
Fonctions « analogReference », « analogRead » .....	16
3.10 – Fonctions particulières de gestion des E/S.....	16
Fonctions « tone », « notone » .....	16
Fonctions « pulseIn », « shiftOut » .....	17
3.11 – Fonctions de manipulation de bits .....	17
Fonction « lowByte ».....	17
Fonction « highByte », « bitRead », « bitWrite », « bitSet », « bitClear » .....	18
3.12 – Fonctions de gestion du port série asynchrone .....	18
Fonction « Serial.begin ».....	18
Fonctions« Serial.end », « Serial.available », « Serial.read », « Serial.flush », « Serial.write »..	19
Fonction « Serial.print » .....	20
3.13 – Bibliothèques.....	20
<b>4 – DEVELOPPEMENT D'UN PROJET ARDUINO.....</b>	<b>21</b>

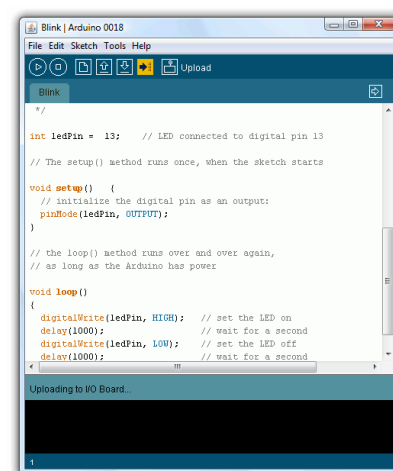
# 1 – PRESENTATION

La carte de développement Arduino est une **plateforme matérielle et logicielle de développement** d'applications embarquées.

Côté matériel, elle se compose d'une **carte électronique** basée autour d'un **microcontrôleur**(ATMEL AVR) comportant un certain nombre d'entrées et de sorties (les ports) permettant la connexion de capteurs, ou d'actionneurs.



Le logiciel de programmation des modules Arduino est une **application Java**, libre et multi-plateformes, servant **d'éditeur de code et de compilateur**, et qui peut transférer le firmware et le programme au travers de la liaison USB. Le langage de programmation utilisé est un **mélange de C et de C++**, restreint et adapté aux possibilités de la carte.



Les principaux avantages du système Arduino sont :

- **Matériel peu onéreux** : Les cartes Arduino sont relativement peu coûteuses comparativement aux autres plateformes de développement. La moins chère des versions du module Arduino peut être assemblée à la main, et même les cartes Arduino pré-assemblées coûtent moins de 25 €.
- **Matériel Open source et extensible** : Les schémas des modules sont publiés sous une licence Creative Commons, et il est possible de réaliser nos propres versions des cartes Arduino, en les complétant et en les améliorant.
- **Logiciel gratuit et Open Source**: Le logiciel Arduino et le langage Arduino sont publiés sous licence open source, disponible pour être complété par des programmeurs expérimentés. Le langage peut être aussi étendu à l'aide de bibliothèques C++.
- **Logiciel multi-plateforme** : Le logiciel Arduino, écrit en Java, tourne sous les systèmes d'exploitation Windows, Macintosh et Linux.
- **Modules « shield »** : Cartes supplémentaires se connectant sur le module Arduino pour augmenter les possibilités : afficheur graphique couleur, interface ethernet, GPS, etc...

## 2 – LA CARTE ARDUINO UNO

### 2.1 – Présentation

Il existe plusieurs types de cartes Arduino. La carte Arduino Uno est une des versions majeures des cartes Arduino. Il s'agit d'une carte au format « standard » Arduino c'est-à-dire environ 52 mm sur 65 mm.



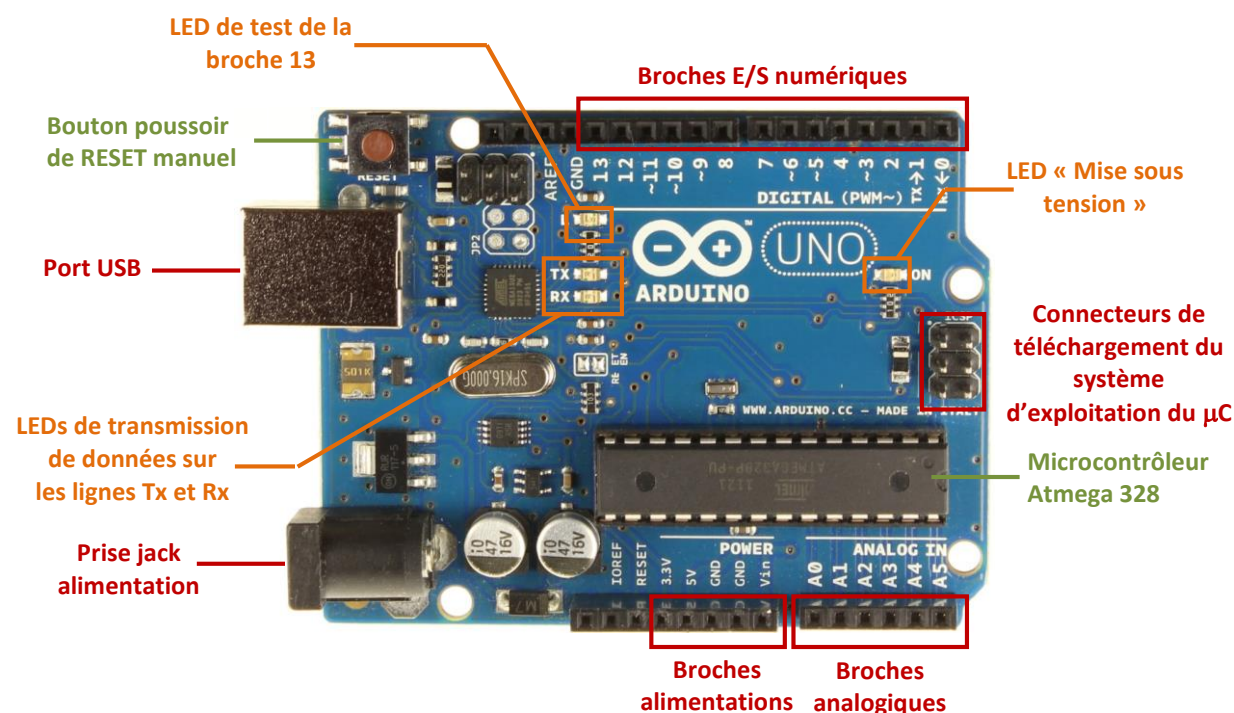
Deux rangées de connecteurs, situées de part et d'autre de la carte, permettent la connexion des composants extérieurs ou des modules shields, au format Arduino, il est possible d'enficher directement ces derniers dans la carte de base, sur plusieurs niveaux si nécessaire.



Elle est équipée d'un microcontrôleur **Atmel ATmega 328**.

(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)
VCC	7	22	GND
GND	8	21	AREF
(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC
(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)

### 2.2 – Différents éléments de la carte Arduino Uno





## 2.3 – Principales caractéristiques

Microcontrôleur	Atmega328
Tension de fonctionnement	5V
Tension d'alimentation (recommandée)	7-12V
Tension d'alimentation (limites)	6-20V
Broches E/S numériques	14 (dont 6 disposent d'une sortie PWM)
Broches Entrées analogiques	6 (utilisables en broches E/S numériques)
Intensité max disponible par broche E/S (5 V)	40 mA (200mA cumulé pour l'ensemble des broches)
Mémoire programme Flash	32 Ko dont 0,5 Ko sont utilisés par le bootloader
Mémoire SRAM (mémoire volatile)	2 Ko
Mémoire EEPROM (mémoire non volatile)	2 Ko
Vitesse d'horloge	2 Ko

## 2.4– Alimentation

La carte Arduino UNO peut être alimentée par le **câble USB**, par un **bloc secteur externe** connecté grâce à une **prise « jack » de 2,1mm** ou bien par un bloc de piles dont le raccordement est réalisé par l'intermédiaire des « **GND** » et « **VIN** » du connecteur d'alimentation (POWER). L'alimentation extérieure doit présenter une tension comprise entre **7 à 12 V**.

La carte génère, par l'intermédiaire de régulateurs intégrés, deux tensions stabilisées : **5 V** et **3,3 V**. Ces deux tensions permettent **l'alimentation des composants électroniques** de la carte Arduino. Etant disponibles sur connecteurs placés sur le pourtour des cartes, elles permettent également **l'alimentation des modules shields**.

## 2.5 –Les entrées/sorties

La carte « Arduino Uno » dispose de **14 E/S numériques** et de **6 entrées analogiques**.

### Entrées/sorties numériques :

Chacune des 14 broches numériques (repérées 0 à 13) peut être utilisée en entrée (input) ou en sortie (output) sous le contrôle du programme. Le sens de fonctionnement pouvant même changer de manière dynamique pendant son exécution.

Elles fonctionnent en logique TTL (0V-5V) ; chacune pouvant fournir (source) ou recevoir un courant maximal de 40 mA et dispose si besoin est d'une résistance interne de 'pull-up'.



Ces lignes partagent leur rôle avec certaines interfaces spécialisées contenues dans le microcontrôleur :

N° E/S	N° ligne de port	Fonction
0	PD0	Rx : Entrée liaison série synchrone
1	PD1	Tx : Sortie liaison série synchrone
2	PD2	INT0 : Entrée interruption externe
3	PD3	INT1 : Entrée interruption externe OC2B : PWM modulation à largeur d'impulsion
4	PD4	T0 : Entrée Timer/compteur 0 XCK : Entrée horloge
5	PD5	T1 : Entrée Timer/compteur1 OC0B : Sortie module PWM modulation à largeur d'impulsion
6	PD6	OC0A : Sortie module PWM modulation à largeur d'impulsion AIN0 : Entrée comparateur analogique
7	PD7	AIN1 : Entrée comparateur analogique
8	PB0	CLKO : Sortie de l'horloge de fonctionnement ICP1 : Entrée de capture Timer/compteur 1
9	PB1	OC1A : Sortie module PWM modulation à largeur d'impulsion
10	PB2	SS : Sélect Slave liaison SPI OC1B : Sortie module PWM modulation à largeur d'impulsion
11	PB3	MOSI : Sortie liaison SPI OC2A : Sortie module PWM modulation à largeur d'impulsion
12	PB4	MISO : Entrée liaison SPI
13	PB5	SCK : Horloge liaison SPI

### Entrées analogiques :

Les six entrées analogiques, repérées **A0 à A5 (PC0 à PC5)**, peuvent admettre toute **tension analogique** comprise entre **0 et 5 V** (par défaut mais cela peut être modifié). Ces entrées analogiques sont gérées par un **convertisseur analogique/numérique de 10 bits**, dont la sortie peut varier de **0 à 1023**.

Les entrées **A4** et **A5** peuvent également être utilisées respectivement comme la ligne de donnée **SDA** et la ligne d'horloge **SCL** de l'**interface série I2C**.

## 2.6–Les mémoires

Le microcontrôleur ATmega 328 dispose de **32 ko de mémoire de programme Flash**. Il contient aussi de **2 ko de mémoire vive (SRAM)**. Cette mémoire est généralement utilisée pour stocker les résultats temporaires lors de calculs. Elle peut être lue et écrite à tout instant par le microcontrôleur mais son contenu est perdu dès que la n'est plus alimentée. L'ATmega 328 dispose également **1ko mémoire EEPROM**. Le contenu de cette mémoire est accessible grâce aux fonctions de la librairie « EEPROM ».



## 2.7 – Communication

La carte « Arduino Uno » a de nombreuses possibilités de communications avec l'extérieur. L'Atmega328 possède une **interface de communication série** UART accessible, grâce aux **broches numériques 0 (Rx)** et **1 (Tx)**. D'autre part elle supporte le **bus I2C** accessible, grâce aux **broches analogiques 4 (SDA)** et **5 (SCL)** et la liaison série synchrone **SPI** grâce aux **broches numériques 10 (SS), 11 (MOSI), 12 (MISO) et 13 (SCX)**.

## 2.8 – Reset

A la **mise sous tension un reset automatique** permet au programme contenu en mémoire du microcontrôleur de **démarrer automatiquement** dès que la carte Arduino est alimentée.

La carte « Arduino Uno » est également équipée d'un **bouton poussoir de reset manuel**. Un appui sur celui-ci permet de relancer l'exécution d'un programme si nécessaire, soit parce qu'il s'est « planté » soit tout simplement parce que l'on souhaite le faire repartir de son début.

## 2.9 – Protection de surintensité USB

Par mesure de sécurité pour l'ordinateur auquel sera relié l'Arduino, un **fusible réarmable** ou « **Polyfuse** » est présent sur la connexion d'alimentation 5 V de la prise USB. Toute consommation **supérieure à 500 mA**, provoque le **déclenchement de ce fusible**, protégeant ainsi le port USB de l'ordinateur auquel la carte est reliée. Le fusible étant de type **réarmable**, il retrouvera son état normal quelques secondes après que la consommation excessive aura cessée.



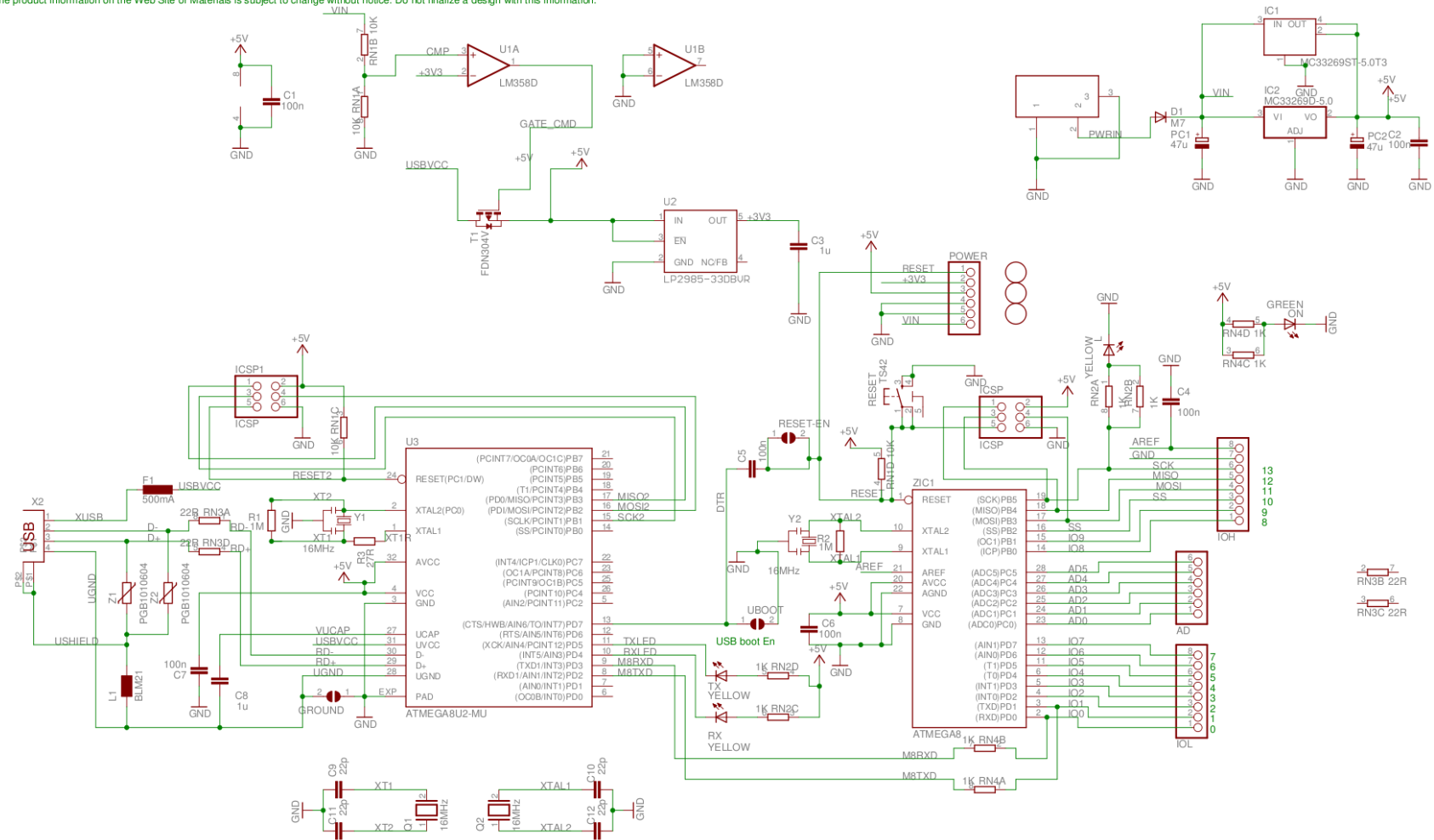


## 2.10 – Schéma structurel de la carte « Arduino Uno »

### Arduino™ UNO Reference Design

Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS". Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.





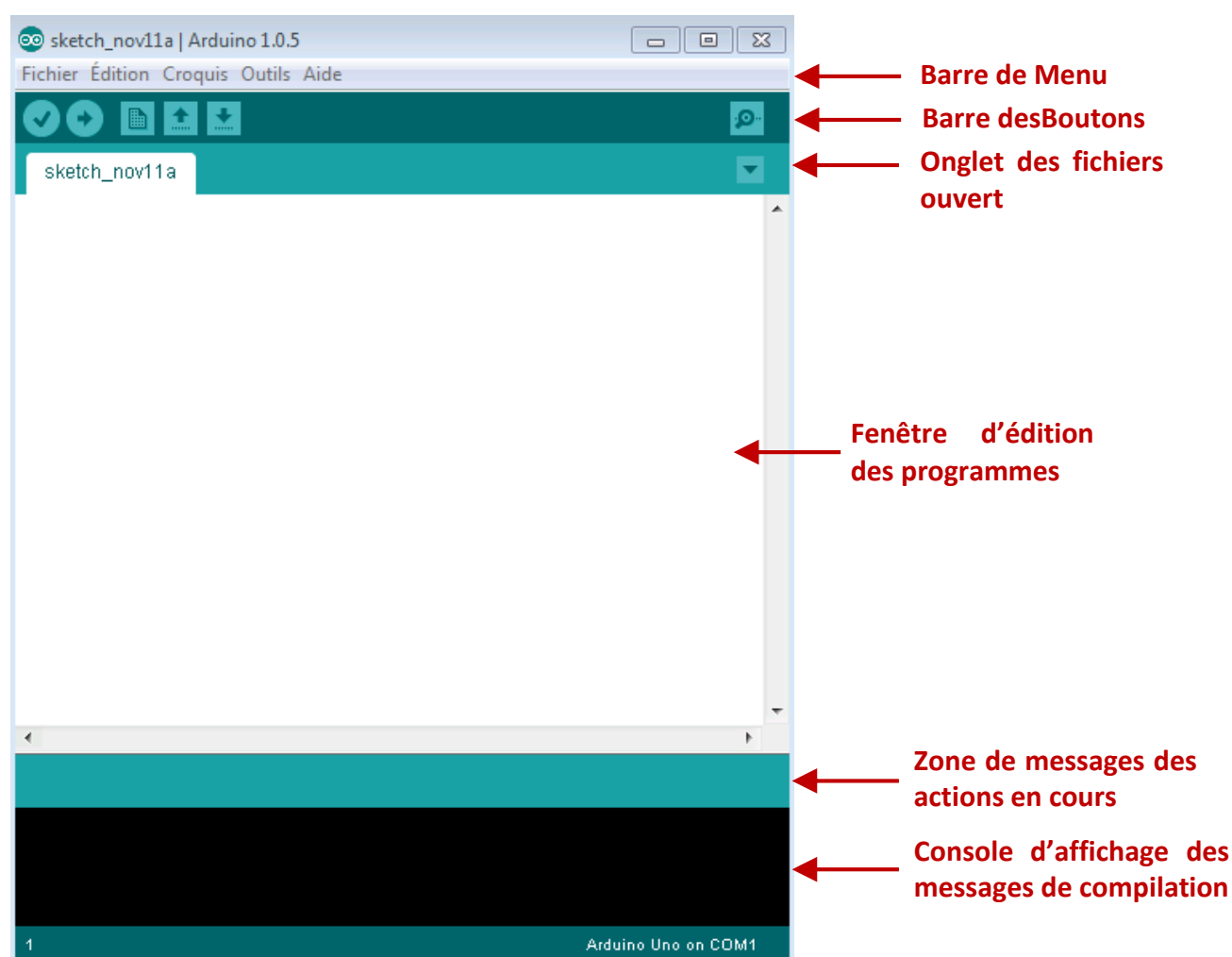
## 3 – PROGRAMMATION DE LA CARTE ARDUINO

### 3.1 – Interface de programmation

Le logiciel Arduino a pour fonctions principales :

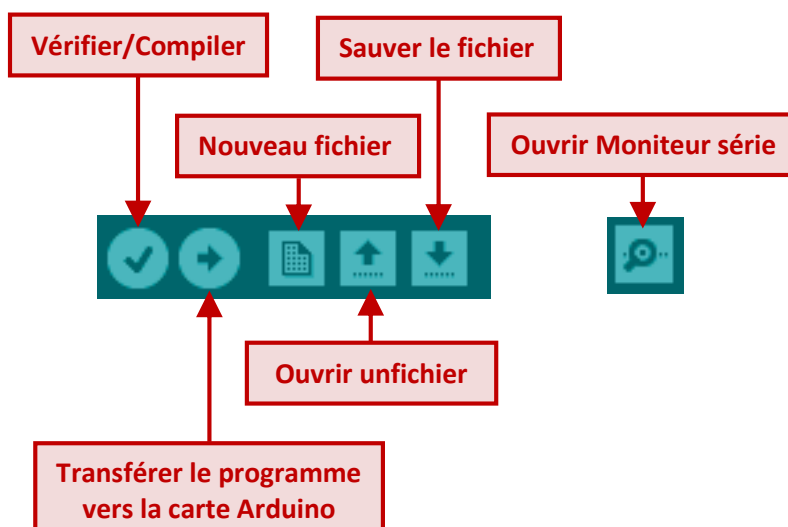
- de pouvoir écrire et compiler des programmes pour la carte Arduino ;
- de se connecter avec la carte Arduino pour y transférer les programmes ;
- de communiquer avec la carte Arduino.

Pour ouvrir l'IDE d'Arduino, il faut utiliser l'icône :





La « **Barre de Boutons** » qui donne un accès direct aux fonctions essentielles du logiciel. Les différents boutons sont :



### 3.2 – Structure d'un programme

Un programme ou croquis (sketch) destiné à une carte Arduino est constitué de 3 parties :

Zone de définition des constantes ou des variables ou d'inclusion des bibliothèques

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

```
int ledPin = 13; // LED connectée à la broche n°13
```

```
void setup()
{
  pinMode(ledPin, OUTPUT); // Definit la broche n°13 comme une sortie
}
```

```
void loop()
{
  digitalWrite(ledPin, HIGH); // Met la sortie 13 au NL1 (diode allumée)
  delay(3000); // Attendre 3 s
  digitalWrite(ledPin, LOW); // Met la sortie 13 au NL0 (diode éteinte)
  delay(1000); // Attendre 1 s
}
```

Fonction `setup()` qui contient les instructions d'initialisation

Fonction `loop()` qui contient les instructions du programme



La première partie permet de **définir les constantes** et **les variables en déclarant leur type**. Elle permet également l'inclusion des bibliothèques utilisées dans le programme au moyen de **#include**.

La fonction **setup()** contient les **instructions d'initialisation ou de configuration des ressources** de la carte comme par exemple, la configuration en entrée ou sorties des broches d'E/S numériques, la définition de la vitesse de communication de l'interface série, etc.. Cette fonction **n'est exécutée qu'une seule fois** juste après le lancement du programme.

La fonction **loop()** contient les **instructions du programme** à proprement parlé. Cette fonction sera **répétée indéfiniment** tant que la carte Arduino restera sous tension.

### Commentaires

Pour placer des commentaires sur une ligne unique ou en fin de ligne, il faut utiliser la syntaxe suivante :

```
// Cette ligne est un commentaire sur UNE SEULE ligne
```

Pour placer des commentaires sur plusieurs lignes :

```
/* Commentaire, sur PLUSIEURS lignes qui sera ignoré par le programme,  
mais pas par celui qui lit le code  
*/
```

## 3.3 – Données, variables et constantes

Les différents types utilisés avec la programmation Arduino sont :

Nom	Description	Exemples
<b>boolean</b>	Donnée logique (true ou false) sur 8 bits	<b>boolean</b> marche = true ;
<b>byte</b>	Octet	<b>byte</b> a = 145 ; <b>byte</b> a = B10010 ;
<b>int</b>	Entier signé sur 16 bits	<b>int</b> b = -2896
<b>unsigned int</b>	Entier non signé sur 16 bits	<b>unsigned int</b> c = 45768
<b>long</b>	Entier signé sur 32 bits	<b>long</b> d = 12345678
<b>unsigned long</b>	Entier non signé sur 32 bits	<b>unsigned long</b> e = 418755889
<b>float</b>	Nombre à virgule flottant sur 32 bits	<b>float</b> f = 2.5891
<b>char</b>	Caractère sur 8 bits. Seuls les caractères ayant pour code ASCII une valeur 0 à 127 sont définis	<b>char</b> lettreA = 'A' ; <b>char</b> lettreA = 65 ;
<b>unsigned char</b>	Caractère sur 8 bits. Caractères du code ASCII étendu (valeur 0 à 255)	<b>unsigned char</b> B = 66

Une constante peut être définie au moyen de **const** ou de **#define** :

**Exemples**

```
#define N 2//Toute variable N rencontrée dans le programme sera remplacée par la valeur 2
const byte N=2 // N est constante de type « byte » et de valeur 2
```

Un certain nombre de noms de constantes sont prédéfinis dans le langage « Arduino » :

Nom	Description
<b>true</b>	Donnée binaire égale à 1 ou donnée numérique différente de 0
<b>false</b>	Donnée binaire ou numérique différente de 0
<b>HIGH</b>	Génération d'un niveau de tension haut sur une des sorties numériques
<b>LOW</b>	Génération d'un niveau de tension bas sur une des sorties numériques
<b>INPUT</b>	Configuration d'une broche numérique en entrée
<b>OUTPUT</b>	Configuration d'une broche numérique en sortie

**3.4 –Les opérateurs****Opérateurs arithmétiques**

+	Addition	$c = a + b ;$
-	Soustraction	$c = a - b ;$
*	Multiplication	$c = a * b ;$
/	Division entière	$c = a / b ;$
%	Reste de la division entière	$c = a \% b ;$

**Affectations**

=	Affectation ordinaire	$a = b ;$
+=	Additionner de	$a += b \Rightarrow a = a + b ;$
-=	Soustraire de	$a -= b \Rightarrow a = a - b ;$
*=	Multiplier par	$a *= b \Rightarrow a = a * b ;$
/=	Diviser par	$a /= b \Rightarrow a = a / b ;$
%=	Reste	$a \% = b \Rightarrow a = a \% b ;$
—	Soustraire de 1 (Décrémentement)	$a \text{ —} \Rightarrow a = a - 1 ;$
++	Additionner 1 (Incrémentement)	$a ++ \Rightarrow a = a + 1$

**Opérateurs binaires**

&	ET	$c = a \& b$
	OU	$c = a   b$
^	OU exclusif	$c = a \wedge b$
~	NON	$b = \sim a$
>>	Décalage à droite des bits	$c = a << b$ (a décalé b fois à droite)
<<	Décalage à gauche des bits	$c = a >> b$ (a décalé b fois à gauche)

**Tests**



&&	ET logique	if (a && b)
	OU Logique	if (a    b)
==	Egal à	if (a == b)
!=	Différente de	if (a != b)
>	Supérieur à	if (a > b)
<	Inférieur à	if (a < b)
>=	Supérieur ou égal à	if (a >= b)
<=	Inférieur ou égal à	if (a <= b)

### 3.5 – Fonctions mathématiques

#### Fonction « min »

`min(x,y) ;`

Cette fonction retourne la **valeur la plus petite** entre les 2 variables x et y

#### Fonction « max »

`max(x,y) ;`

Cette fonction retourne la **valeur la plus grande** entre les 2 variables x et y

#### Fonction « constrain »

`constrain(x,a,b) ;`

Cette fonction **impose des valeurs** pour la variable x **comprises entre les bornes a et b**. Cette fonction peut être utilisée pour limiter la tension fournie par un capteur.

#### Fonction « abs »

`abs(x) ;`

Cette fonction retourne la **valeur absolue** de la variable x.

#### Fonction « sqrt »

`sqrt(x) ;`

Cette fonction retourne la racine carrée de x.

#### Fonction « map »



```
map(x,min,max,newmin,newmax) ;
```

Cette fonction permet de réaliser une **translation de la plage de variation** de la variable x.

La variable x qui évolue normalement de la valeur min à la valeur max, évoluera après exécution de la fonction map, de la valeur newmin à la valeur newmax.

Cette fonction est très utilisée pour le traitement des informations délivrées par un capteur.

### 3.6 – Fonctions de gestion du temps

#### Fonction « delay »

```
delay(tempo) ;
```

Cette fonction **génère une pause** dont la durée est égale à **tempo × 1 ms**.

La variable tempo doit être de type **unsigned long** ce qui peut autoriser des temporisations qui peuvent être très longues.

```
delay(1000) ; // Pause de 1 s
```

#### Fonction « delayMicroseconds »

```
delayMicroseconds(tempo) ;
```

Cette fonction **génère une pause** dont la durée est égale à **tempo × 1 µs**.

La variable tempo doit être de type **unsigned int**.

⚠ La durée de la pause n'est assurée que des **valeurs supérieures à 3 µs**.

```
delay(100) ; // Pause de 100µs
```

### 3.7 – Fonctions de gestion des E/S numériques

#### Fonction « pinMode »

```
pinMode(Numéro broche, Sens de fonctionnement) ;
```

Cette fonction permet de **configurer le sens de fonctionnement**, entrée ou sortie, de la broche sélectionnée.

```
pinMode(12,OUTPUT) ;// La broche 12 est configurée comme une sortie
```

#### Fonction « digitalWrite »



`digitalWrite`(Numéro broche, Niveau logique) ;

Cette fonction permet **d'imposer un niveau logique haut ou bas** sur la sortie numérique sélectionnée.

`digitalWrite`(12, `HIGH`) ;// La sortie 12 est placée au niveau logique haut (NL1)

#### Fonction « `digitalRead` »

valeur = `digitalRead`(Numéro broche) ;

Cette fonction permet **de lire le niveau logique** sur l'entrée numérique sélectionnée. Cette fonction ne retourne que deux valeur HIGH ou LOW.

etat = `digitalRead`(11) ;// La variable etat est égale au niveau logique de l'entrée 11

### 3.8 – Fonction de gestion des sorties analogiques

Les µC qui équipent les cartes Arduino sont capables de générer des signaux à **Modulation à Largeur d'Impulsions (MLI)** ou « **Pulse Width Modulation** » (**PWM**). Il s'agit de signaux logiques rectangulaires de fréquence égale à 490 Hz et à rapport cyclique programmable.

#### Fonction « `analogWrite` »

`analogWrite`(Numéro sortie, Rapport cyclique) ;

Cette fonction permet de **générer**, sur la sortie sélectionnée, **un signal PWM** avec le rapport cyclique désiré. La variable « Rapport cyclique » doit être de **type int**, c'est-à-dire qu'elle peut être comprise entre **0 et 255** pour un rapport cyclique, pour le signal généré, compris entre **0 et 100 %**.

Le signal PWM est généré à partir de l'instant de l'exécution de la fonction « `analogWrite` » et jusqu'à ce qu'une nouvelle instruction « `analogWrite` » ou une instruction « `digitalRead` » ou « `digitalWrite` » soit exécutée sur la même broche.

⚠ La **valeur du rapport cyclique** n'est garantie que des valeurs **supérieures à 10 %**.

`analogWrite`(2,64) ;// Signal PWM de rapport cyclique de 25 % sur la sortie A2

### 3.9 – Fonctions de gestion des entrées analogiques

Les entrées analogiques sont connectées à un **convertisseur analogique numérique (CAN)** 10 bits. La sortie du CAN génère une donnée égale à **0** lorsque la **tension d'entrée est nulle** et une donnée égale à **1023** lorsque la **tension d'entrée est égale à la tension de référence du CAN**.

#### Fonction « `analogReference` »





**analogReference**(Type) ;

Cette fonction permet **de définir la tension de référence du CAN**.

La variable type peut prendre les valeurs suivantes :

- ☐ **DEFAULT** : La tension de référence est égale à la **tension d'alimentation** de la carte Arduino (5V pour la carte Arduino Uno) ;
- ☐ **INTERNAL** : La tension de référence est égale à 1,1 V pour la carte Arduino Uno ;
- ☐ **EXTERNAL** : La tension de référence est égale à la **tension appliquée sur l'entrée externe AREF** de la carte Arduino.

⚠ La **tension appliquée sur une entrée analogique** ne doit jamais être supérieure à la **tension d'alimentation** de la carte.

#### Fonction « analogRead »

valeur = **analogRead**(Numéro entrée) ;

Cette fonction permet **de lire le résultat de la conversion analogique numérique** de la tension présente sur l'entrée analogique sélectionnée.

La **donnée retournée** étant comprise entre 0 et 1023, doit être de **type int**.

⚠ La **durée** de la conversion est d'environ **100 µs**.

tension = **analogRead**(0) ;// tension est égale au résultat de la CAN de l'entrée A0

### 3.10 – Fonctions particulières de gestion des E/S

#### Fonction « tone »

**tone**(Numéro sortie, Fréquence, Durée) ;

Cette fonction permet de **générer**, sur la sortie sélectionnée, **un signal carré** (rapport cyclique de 50 %) de **fréquence programmable** et pendant la **durée désirée**.

La donnée « **Fréquence** » doit être donnée en **Hz** et la donnée « **Durée** » doit être fixée en **ms**. Si cette durée n'est pas précisée, le signal est généré jusqu'à ce que la fonction « notone » soit exécutée.

#### Fonction « notone »

**notone**(Numéro sortie) ;

Cette fonction permet de **stopper la génération du signal carré** sur la sortie sélectionnée.

#### Fonction « pulseIn »



`durée = pulseIn(Numéro entrée, Type, Delai max) ;`

Cette fonction permet de **mesurer la durée d'une impulsion** sur l'entrée sélectionnée. Le résultat retourné est du type **unsigned long**.

La valeur de la donnée « **Type** » peut être **HIGH** dans le cas d'une **impulsion au NL1** ou **LOW** d'une **impulsion au NL0**.

La variable « **Delai max** » permet de définir le **délai d'attente de l'impulsion**. **Lorsque ce délai est dépassé** et qu'aucune impulsion ne s'est pas produite, la fonction retourne une **valeur nulle**. Cette donnée est du type **unsigned long** et est exprimé en  $\mu s$ . Si cette donnée n'est pas précisée, la valeur **par défaut** sera de **1 s**.

⚠ Le résultat fourni n'est considéré comme correct que pour des impulsions de durée comprise entre **10  $\mu s$  et 3 min**.

#### Fonction « `shiftOut` »

`shiftOut(Sortie Donnée, Sortie Horloge, Ordre des bits, Donnée) ;`

Cette fonction permet de **réaliser une liaison synchrone (SPI) de manière logicielle** sur n'importe quelle sortie du microcontrôleur dans le cas où l'interface SPI du microcontrôleur ne puisse pas être utilisée.

La variable « **Sortie Donnée** » permet de sélectionner la **broche de sortie des données** et la variable « **Sortie Horloge** » permet de sélectionner la **broche utilisée pour le signal d'horloge**.

La valeur de la variable « **Ordre des bits** » peut prendre la valeur **MSBFIRST** dans le cas où le **bit de poids fort est transmis le premier** ou **LBSFIRST** si le **bit de poids faible est transmis le premier**.

La variable « **Donnée** » est la **donnée à transmettre**. Il s'agit d'une variable de **type byte** (8 bits).

⚠ Il est plus intéressant tout de même d'utiliser l'interface SPI et les fonctions de la bibliothèque SPI associée.

### 3.11 – Fonctions de manipulation de bits

#### Fonction « `lowByte` »

`resultat = lowByte(Donnée) ;`

Cette fonction permet d'**extraire les 8 bits de poids faible** de la variable « **Donnée** ».

La **valeur retournée** est de **type byte** alors que la variable « **Donnée** » peut être de **n'importe quel type** (donc n'importe quelle taille).

#### Fonction « `highByte` »



```
resultat = lowByte(Donnée) ;
```

Cette fonction permet d'**extraire le deuxième octet en partant de la droite** de la variable « Donnée ».

La **valeur retournée** est de **type byte** alors que la variable « Donnée » peut être de **n'importe quel type**.

#### Fonction « bitRead »

```
resultat = bitRead(Donnée,n) ;
```

Cette fonction permet de **lire le n<sup>ième</sup> bit de la variable « Donnée »**.

#### Fonction « bitWrite »

```
bitWrite(Donnée,n,Niveau) ;
```

Cette fonction permet d'**imposer un niveau logique sur le n<sup>ième</sup> bit de la variable « Donnée »**.

#### Fonction « bitSet »

```
bitSet(Donnée,n) ;
```

Cette fonction permet d'**imposer un 1 logique sur le n<sup>ième</sup> bit de la variable « Donnée »**.

#### Fonction « bitClear »

```
bitClear(Donnée,n) ;
```

Cette fonction permet d'**imposer un 0 logique sur le n<sup>ième</sup> bit de la variable « Donnée »**.

### 3.12 – Fonctions de gestion du port série asynchrone

La carte Arduino Uno dispose d'un port série asynchrone accessible via les E/S numériques 0 (ligne Rx) et 1 (Ligne Tx).

#### Fonction « Serial.begin »

```
Serial.begin(Vitesse) ;
```

Cette fonction qui doit être **appelée au moins une fois**, généralement dans la fonction setup(), permet de **définir la vitesse** utilisée par la liaison série.

La valeur prise par la variable « **Vitesse** » doit être une des **vitesse définies par la norme RS232**.

#### Fonction « Serial.end »



**Serial.end()** ;

Cette fonction permet de **désactiver la liaison série asynchrone** et donc de libérer les broches numériques 0 et 1.

Le port série asynchrone des microcontrôleurs qui équipent les cartes Arduino disposent d'une **mémoire tampon** capable de **mémoriser jusqu'à 128 caractères reçus**.

#### Fonction « Serial.available »

nombre = **Serial.available()** ;

Cette fonction permet de **connaître le nombre de caractères reçus** et donc contenus dans la mémoire tampon en attente de lecture par le programme. La variable « nombre » est de **type int**.

#### Fonction « Serial.read »

caractère = **Serial.read()** ;

Cette fonction permet de **lire le premier caractère disponible dans la mémoire tampon de réception**. La variable « caractère » est de **type int**. La valeur retournée est -1 si aucun caractère n'a été reçu (mémoire tampon vide).

#### Fonction « Serial.flush »

**Serial.flush()** ;

Cette fonction permet de **vider la mémoire tampon**.

#### Fonction « Serial.write »

**Serial.write**(Donnée) ;

Cette fonction permet d'**émettre une ou plusieurs données sur la liaison série** sous la forme binaire brute. Si la donnée est de type chaîne de caractères, les caractères sont transmis, sous le format ASCII, les uns après les autres.

#### Fonction « Serial.print »



**Serial.print**(Donnée, Format) ;

Cette fonction permet d'émettre une ou plusieurs données sur la liaison série en précisant le codage utilisé.

Si la variable « **Format** » n'est pas précisée, une donnée numérique sera considérée comme décimale, une chaîne de caractères sera transmise tel quelle et une donnée à virgule flottante sera transmise uniquement avec deux décimales.

Sinon la variable « **Format** » permet de préciser le codage ou la base de numération pour les données numériques. Elle peut prendre une des valeurs suivantes : **BIN** (binaire), **OCT** (Octal), **HEX** (hexadécimal), **DEC** (Décimal) ou **BYTE** (codage ASCII). Pour les données à virgule flottante, la variable « **Format** » peut prendre une valeur numérique comprise entre 0 et 7 correspondant au nombre de décimales à transmettre.

**Serial.print**(75,BIN) ; // Transmission de la donnée 100101

**Serial.print**(75,HEX) ; // Transmission de la donnée 4B

**Serial.print**(75,BYTE) ; // Transmission du caractère K

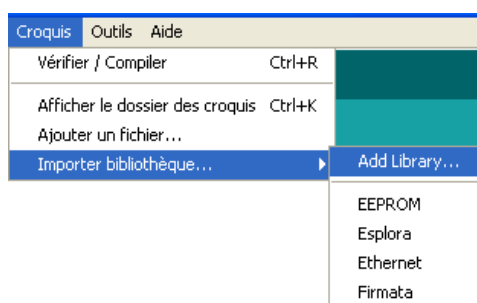
**Serial.print**(1.23456,0) ; // Transmission de la valeur 1

**Serial.print**(1.23456,2) ; // Transmission de la valeur 1.23456

### 3.13 – Bibliothèques

Une bibliothèque est un ensemble de fonctions utilitaires mises à disposition des utilisateurs de l'environnement Arduino. Les fonctions sont regroupées en fonction de leur appartenance à un même domaine conceptuel (mathématique, graphique, tris, etc).

L'IDE Arduino comporte par défaut plusieurs bibliothèques externes. Pour les importer dans votre programme, vous devez utiliser le menu suivant :



L'instruction suivante sera alors ajoutée au début de votre programme : `#include <la_bibliothèque.h>`

#### Exemple

```
#include <SPI.h> // Importation de la bibliothèque SPI
```

Les bibliothèques fournies par l'IDE Arduino sont :



- **EEPROM** : lecture et écriture de données dans une mémoire EEPROM ;
- **Ethernet** : connexion et transmission de données en utilisant le Shield Ethernet ;
- **Arduino Firmata** : applications utilisant un protocole série ;
- **LiquidCrystal** : contrôle d'afficheurs à cristaux liquides (LCD) ;
- **SD** : lecture et écriture de données sur des cartes SD ;
- **Servo** : contrôle des servomoteurs.
- **SPI** : transmission de données par protocole de communication SPI (Serial Peripheral Interface) ;
- **SoftwareSerial** : communication série supplémentaire sur l'E/S numériques ;
- **Stepper** : commande de moteurs « pas à pas » ;
- **Wire** : communication TWI/I2C.


D'autres librairies sont disponibles en téléchargement à l'adresse suivante :

<http://www.arduino.cc/en/Reference/Libraries>

Pour installer ces librairies, il faut décompresser le fichier téléchargé et le stocker dans un répertoire appelé « **libraries** » situé dans le répertoire « **Arduino** » créé, dans le dossier personnel, au premier lancement de l'IDE Arduino.

## 4 – DEVELOPPEMENT D'UN PROJET SOUS ARDUINO

Le développement d'un projet sous Arduino nécessite les phases suivantes :

1. Lancer l'IDE Arduino en cliquant sur l'icône .

2. Editer le programme à partir de l'IDE Arduino ;

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

int ledPin = 13;          // LED connectée à la broche n°13

void setup()
{
  pinMode(ledPin, OUTPUT); // Definit la broche n°13 comme une sortie
}

void loop()
{
  digitalWrite(ledPin, HIGH); // Met la sortie 13 au NL1 (diode allumée)
  delay(3000);                // Attendre 3 s
  digitalWrite(ledPin, LOW);  // Met la sortie 13 au NL0 (diode éteinte)
  delay(1000);                // Attendre 1 s
}
```

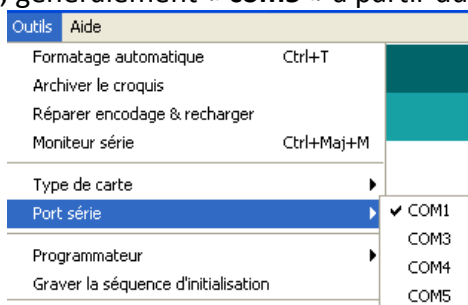
3. Vérifier et compiler le code à l'aide du bouton «  ifier ».



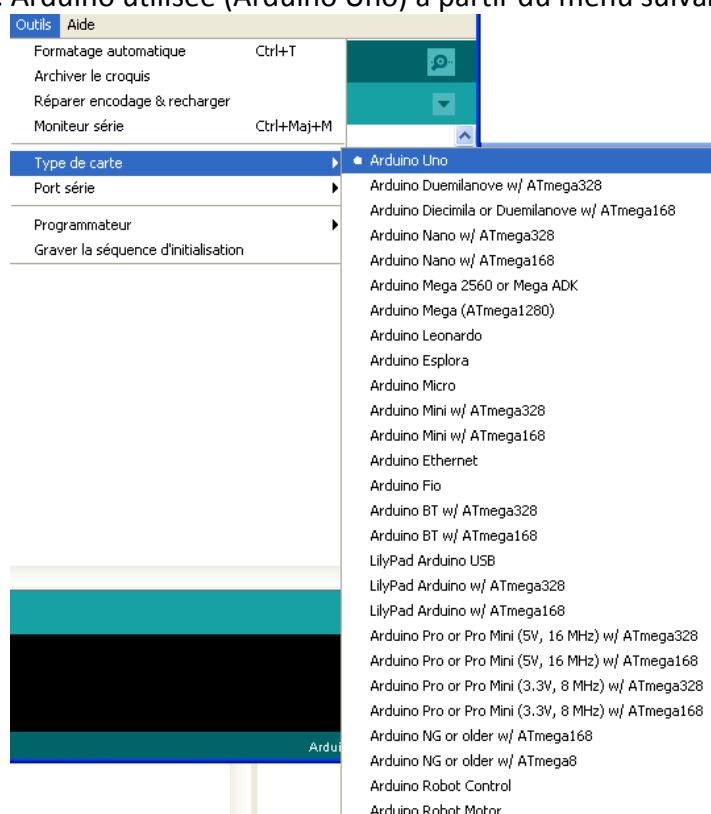
La vérification et la compilation est correcte si aucun message d'erreur n'apparaît, le message « Compilation terminée » est notifié dans la « zone des messages des actions en cours » et aucun message d'erreur n'apparaît dans le console des « messages de compilation ».

```
Compilation terminée.  
  
Taille binaire du croquis : 1 084 octets (d'un max de 32 256 octets)
```

4. Sélectionner le bon port série, généralement « com5 » à partir du menu suivant :



5. Sélectionner la carte Arduino utilisée (Arduino Uno) à partir du menu suivant :



6. Charger le programme dans la carte Arduino en cliquant sur le bouton  « Téléverser ».

7. Vérifier le fonctionnement du programme sur la carte Arduino.