Pour pouvoir programmer notre carte, il nous faut trois choses :

- Un ordinateur
- Une carte Arduino
- Et connaitre le langage Arduino

C'est ce dernier point qu'il nous faut acquérir. Le but même de ce chapitre est de vous apprendre à programmer avec le langage Arduino.

Le langage Arduino est très proche du C et du C++.

I - La syntaxe du langage

La syntaxe d'un langage de programmation est l'ensemble des règles d'écritures liées à ce langage. On va donc voir dans ce sous chapitre les règles qui régissent l'écriture du langage Arduino.

Le code minimal

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

Code : C

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

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

La fonction

Dans ce code se trouvent deux fonctions. Les fonctions sont en fait des portions de code.

Cette fonction setup() est appelée une seule fois lorsque le programme commence. C'est pourquoi c'est dans cette fonction que l'on va écrire le code qui n'a besoin d'être exécuté une seule fois. On appelle cette fonction : "fonction d'initialisation ". On y retrouvera la mise en place des différentes sorties et quelques autres réglages. C'est un peu le check-up de démarrage.

Une fois que l'on a initialisé le programme il faut ensuite créer son "coeur", autrement dit le programme en lui même.

C'est donc dans cette fonction loop() où l'on va écrire le contenu du programme. Il faut savoir que cette fonction est appelée en permanence, c'est-à-dire qu'elle est exécutée une fois, puis lorsque son exécution est terminée, on la ré-exécute et encore et encore. On parle de boucle infinie.

A titre informatif, on n'est pas obligé d'écrire quelque chose dans ces deux fonctions. En revanche, il est obligatoire de les écrire, même si elles ne contiennent aucun code !

II - Les instructions

Les instructions sont des lignes de code qui disent au programme : "fait ceci, fait cela, ..." C'est tout bête mais très puissant car c'est ce qui va orchestrer notre programme.

Les points virgules

Les points virgules terminent les instructions. Si par exemple je dis dans mon programme : "appelle la fonction fairedemitour" je dois mettre un point virgule après l'appel de cette fonction.

Les accolades

Les accolades sont les "conteneurs" du code du programme. Elles sont propres aux fonctions, aux conditions et aux boucles.
Les instructions du programme sont écrites à l'intérieur de ces accolades.

Les commentaires

Pour finir, on va voir ce qu'est un commentaire. J'en ai déjà mis dans les exemples de codes. Ce sont des lignes de codes qui seront ignorées par le programme. Elles ne servent en rien lors de l'exécution du programme.

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

/*cette ligne est un commentaire, sur PLUSIEURS lignes
qui sera ignoré par le programme, mais pas par celui qui li le code
;) */
```

Les accents

Il est formellement interdit de mettre des accents en programmation. Sauf dans les commentaires.

III - Les variables

Imaginons que vous avez connecté un bouton poussoir sur une broche de votre carte Arduino.
Comment allez-vous stocker l'état du bouton (appuyé ou éteint) ?

Une variable est un nombre . Ce nombre est stocké dans un espace de la mémoire vive (RAM) du microcontrôleur. La manière qui permet de les stocker est semblable à celle utilisée pour ranger des chaussures : dans un casier numéroté.

Ce nombre a la particularité de changer de valeur.

Le symbole "=>" signifiant : "est contenu dans..."

Le nom de variable accepte quasiment tous les caractères sauf :

- . (le point)
- , (la virgule)
- é,à,ç,è (les accents)

Définir une variable

Si on donne un nombre à notre programme, il ne sait pas si c'est une variable ou pas. Il faut le lui indiquer. Pour cela, on donne un type aux variables. Oui, car il existe plusieurs types de variables ! Par exemple la variable "x" vaut 4 :

```
x = 4 ;
```

Et bien ce code ne fonctionnerait pas car il ne suffit pas ! En effet, il existe une multitude de nombres : les nombres entiers, les nombres décimaux, ... C'est pour cela qu'il faut assigner une variable à un type.

Voilà les types de variables les plus répandus :

Type	Quel nombre il stocke ?	Valeurs max du nombre stocké	Nombre sur x bits	Nombre d'octets
Int	entier	-32 768 à +32 767	16	2
Long	entier	-2 147 483 648 à +2 147 483 647	32	4
Char	entier	-128 à +127	8	1
Flat	décimale	$-3{,}4\times10^{38}$ à $+3{,}4\times10^{38}$	32	4
double	décimale	$-3{,}4\times10^{38}$ à $+3{,}4\times10^{38}$	32	4

Par exemple, si notre variable "x" ne prend que des valeurs décimales, on utilisera les types int , long , ou char . Si maintenant la variable "x" ne dépasse pas la valeur 64 ou 87, alors on utilisera le type char .

```
char  x = 0 ;
```

Si à présent notre variable "x" ne prend jamais une valeur négative (-20, -78, ...), alors on utilisera un type non-signé. C'est à dire, dans notre cas, un char dont la valeur n'est plus de -128 à +127, mais de 0 à 255.

Type	Quel nombre il stocke ?	Valeurs max du nombre stocké	Nombre sur x bits	Nombre d'octets
unsigned char	Entier positif	0 à 255	8	1
unsigned int	Entier positif	0 à 65 535	16	2
unsigned long	Entier positif	0 à 4 294 967 295	32	4

Une des particularités du langage Arduino est qu'il accepte un nombre plus important de types de variables. Je vous les liste dans ce tableau :

Type	Quel nombre il stocke ?	Valeurs max du nombre stocké	Nombre sur x bits	Nombre d'octets
byte	Entier positif	0 à 255	8	1
word	Entier positif	0 à 65 535	16	2
boolean	Entier positif	0 à 1	1	1

Les variables booléennes

Les variables booléennes sont des variables qui ne peuvent prendre que deux valeurs : ou VRAI ou FAUX. Elles sont utilisées notamment dans les boucles et les conditions.

Une variable booléenne peut être définie de plusieurs manières :

```
boolean variable =  FALSE; //variable est fausse car elle vaut FALSE, du terme anglais "faux"
boolean variable =  TRUE; //variable est vraie car elle vaut TRUE, du terme anglais "vrai"
```

ou

```
int  variable = 0 ; //variable est fausse car elle vaut 0
int  variable = 1 ; //variable est vraie car elle vaut 1
int  variable = 42 ; //variable est vraie car sa valeur est différente de 0
```

ou plus particulièrement dans le langage Arduino

```
int  variable =  LOW; //variable est à l'état logique bas (= traduction de "low"), donc 0
int  variable =  HIGH; //variable est à l'état logique haut (= traduction de "high"), donc 1
```

IV – Les opérations simples

L'addition

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

Faisons maintenant une addition de variables :

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

Le principe est le même pour ***la soustraction***, ***la multiplication*** et ***la division.***

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

L'incrémentation et la décrémentation

C'est une simple opération d'addition.

```
var = 0 ;
var++ ; //c'est cette ligne de code qui nous intéresse qui revient à écrire var=var+1
var +=6 ; // ici var=var+6
var-- ; / /ici var=var-1
var -=6; / /ici var=var-6
var /=5 ; / /ici var=var/5
var %=6 ; / /ici var=var % 6
```

L'opération de bascule

```
boolean x = 0 ; //on définit une variable x qui ne peut prendre que la valeur 0 ou 1 (vraie ou fausse)
 x = 1 -  x;

ou

x= !x
```

V – Les conditions

C'est un choix que l'on fait entre plusieurs propositions. En informatique, les conditions servent à tester des variables.

Quelques symboles

Symbole	A quoi il sert	Signifiaction
==	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...

If...else

```
if(/* contenu de la condition à tester */ )
{
//instructions à exécuter si la condition est vraie
}
else
{
// instructions à exécuter si la condition est fausse
}
```

switch

Le switch , comme son nom l'indique, va tester la variable jusqu'à la fin des valeurs qu'on lui aura données. Voici comment cela se présente :

```
int  options_voiture = 0 ;
switch  (options_voiture)
{
case 0 :
//il n'y a pas d'options dans la voiture
break ;
case 1 :
//la voiture a l'option GPS
break ;
case 2 :
//la voiture a l'option climatisation
break ;
default:
//retente ta chance ;-)
break ;
}
```

VI– Les opérateurs logiques

Il existe des opérateurs qui vont nous permettre de tester les conditions.

Opérateur	Signification
&&	…ET…
||	…OU…
!	NON

ET

```
int  prix_voiture = 5500 ;
int  option_GPS =  TRUE;

if (prix_voiture == 5500 &&  option_GPS) /*l'opérateur && lie les deux conditions qui doivent être vraies ensemble pour que la condition soit remplie*/
 {
//j'achète la voiture si la condition précédente est vraie
 }
```

OU

```
if((prix_voiture < 5000) || (prix_voiture == 5500 && option_GPS))
{
//la condition est vraie, donc j'achète la voiture
}
else
{
//la condition est fausse, donc je n'achète pas la voiture
}
```

NON

```
int  prix_voiture = 5500 ;
if (! (prix_voiture < 5000 ))
{
//la condition est vraie, donc j'achète la voiture
 }
```

VII– Les boucles

En programmation, une boucle est une instruction qui permet de répéter un bout de code. Cela va nous permettre de faire se répéter un bout de programme ou un programme entier.
Il existe deux types principaux de boucles :
La boucle conditionnelle , qui teste une condition et qui exécute les instructions qu'elle contient tant que la condition testée est vraie.
La boucle de répétition , qui exécute les instructions qu'elle contient, un nombre de fois prédéterminé.

La boucle while

En anglais, le mot while signifie "tant que". Donc si on lit la ligne :

```
while(position_volet = "ouvert")
        {
        /* instructions */
        }
```

Il faut la lire : "TANT QUE la position du volet est ouvert ", on exécute les instructions entre les accolades.

Prenons un exemple simple, réalisons un compteur !

```
int  compteur = 0 ; //variable compteur
        while (compteur != 5 ) //tant que compteur est différent de 5, on boucle
        {
        compteur++ ; //on incrémente la variable compteur à chaque tour de boucle
         }
```

La boucle do...while

Cette boucle est similaire à la précédente. Mais il y a une différence qui a son importance ! En effet, si on prête attention à la place la condition dans la boucle while , on s'aperçoit qu'elle est testée avant de rentrer dans la boucle.

```
do
{
//les instructions entre ces accolades sont répétées tant que la condition est fausse
}while(/* condition à tester */ )
```

Dans une while , si la condition est vraie dès le départ, on entrera jamais dans cette boucle. A l'inverse, avec une boucle do...while , on entre dans la boucle puis on test la condition.

La boucle for

Voilà une boucle bien particulière. Ce qu'elle va nous permettre de faire est assez simple. Cette boucle est exécutée X fois.
Contrairement aux deux boucles précédentes, on doit lui donner trois paramètres.

```
for(/*initialisation de la variable*/ ; /*condition à laquelle la boucle s'arrête*/ ; /*instruction à exécuter*/ )

for(int compteur = 0; compteur < 5; compteur++)
{
//code à exécuter
}
```

D'abord, on créer la boucle avec le terme for (signifie "pour que"). Ensuite, entre les parenthèses, on doit donner trois paramètres qui sont :

- la création et l'assignation de la variable à une valeur de départ
- suivit de la définition de la condition à tester
- suivit de l'instruction à exécuter

Le langage Arduino n'accepte pas l'absence de la ligne suivante :
int compteur
On est obligé de déclarer la variable que l'on va utiliser (avec son type) dans la boucle for !

La boucle infinie

La boucle infinie est très simple à réaliser, d'autant plus qu'elle est parfois très utile. Il suffit simplement d'utiliser une while et de lui assigner comme condition une valeur qui ne change jamais. En l'occurrence, on met souvent le chiffre 1 ou TRUE.

```
while(1)
{
//instructions à répéter jusqu'à l'infinie
}

while(TRUE)
{
//instructions à répéter jusqu'à l'infinie
}
```

VIII– 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 fait, lorsque l'on va programmer notre carte Arduino, on va écrire notre programme dans des fonctions. Pour l'instant nous n'en connaissons que 2 : setup() et loop() .

En programmation, les fonctions sont "réparties dans deux grandes familles". En effet il existe des fonctions toutes prêtes dans le langage Arduino et d'autres que l'on va devoir créer nous même. C'est ce dernier point qui va nous intéresser.

Fabriquer une fonction

Pour fabriquer une fonction, nous avons besoin de savoir trois choses :

- Quel est le type de la fonction que je souhaite créer ?
- Quel sera son nom ?
- Quel(s) paramètre(s) prendra-t-elle ?

Les fonctions vides

Une fonction qui n'accepte pas de paramètres est une fonction vide.

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

On utilise donc le type void pour dire que la fonction n'aura pas de paramètres.

Les fonctions "typées"

Si on veut créer une fonction qui calcule le résultat d'une opération (ou un calcul plus complexe), il serait bien de pouvoir renvoyer directement le résultat.
Lorsqu'elle sera appelée, la fonction maFonction() va tout simplement retourner la variable résultat.

```
int  calcul = 0 ;
void loop ()
{
calcul = 10 *  maFonction();
}
int maFonction ()
{
int  resultat = 44 ; //déclaration de ma variable résultat
return  resultat;
}
```

Dans la fonction loop() , on fait un calcul avec la valeur que nous retourne la fonction maFonction() .

Les fonctions avec paramètres

Pour comprendre prenons l'exemple ci-dessous :

```
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;
}
```

La fonction récupère dans des variables les paramètres que l'on lui a envoyés. Autrement dit, dans la variable param1, on retrouve la variable x . Dans la variable param2 , on retrouve la variable y .

IX– Les tableaux

On va principalement utiliser des tableaux lorsque l'on aura besoin de stocker des informations sans pour autant créer une variable pour chaque information.

Déclarer un tableau

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.

Voici un exemple pour un tableau de 20 éléments :

```
float  tab[20 ];
```

Ici on choisit float car ce type accepte les nombres à virgule. On aurait pu prendre char.
On peut également créer un tableau vide, la syntaxe est la suivante :

```
float notes[] = {};
```

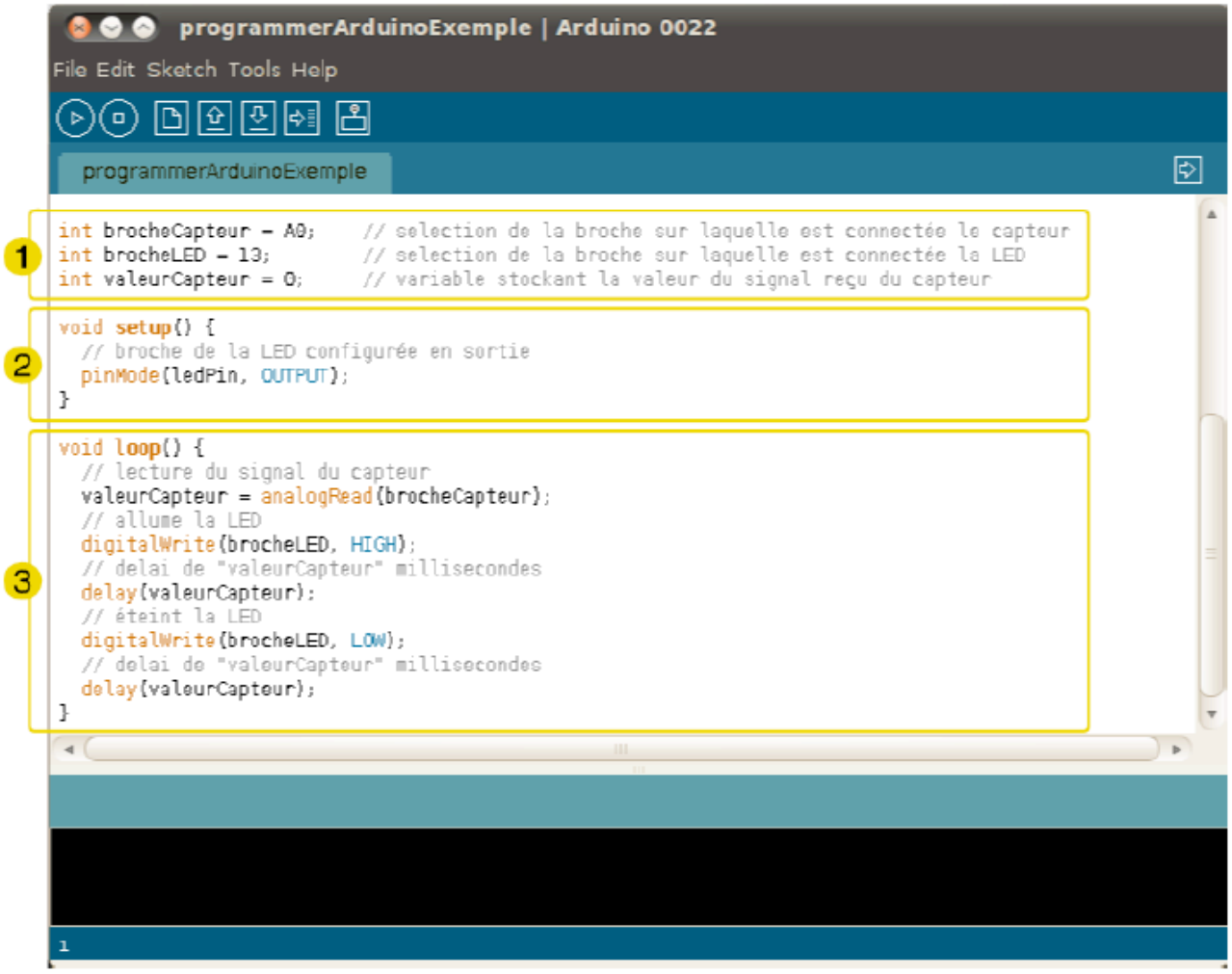
Accéder et modifier une case du tableau

Pour accéder à une case d'un tableau, il suffit de connaitre l'indice de la case que l'on veut accéder. L'indice c'est le numéro de la case qu'on veut lire/écrire. Par exemple, pour lire la valeur de la case 10 (donc indice 9 car on commence à 0):

```
float  tab[20 ]; //notre tableau
float  valeur; //une variable qui contiendra une note
valeur =  tab[9 ] ; //valeur contient le nombre du dixième élément du tableau
```

X– Programmer Arduino

La structure d'un programme



1. la partie déclaration des variables (optionnelle)
2. la partie initialisation et configuration des entrées/sorties : la fonction setup ()
3. la partie principale qui s'exécute en boucle : la fonction loop ()

Entrées/Sorties Numériques

pinMode(broche, mode)

Configure la broche spécifiée pour qu'elle se comporte soit en entrée, soit en sortie.

broche: le numéro de la broche de la carte Arduino dont le mode de fonctionnement (entrée ou sortie) doit être défini.
mode: soit INPUT (entrée en anglais) ou OUTPUT (sortie en anglais)

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

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

digitalWrite(broche, valeur)

Met un niveau logique HIGH (HAUT en anglais) ou LOW (BAS en anglais) sur une broche numérique.

broche: le numéro de la broche de la carte Arduino
valeur : HIGH ou LOW (ou bien 1 ou 0)

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

void setup()
{
  pinMode(ledPin, OUTPUT);      // met la broche utilisée avec la LED en SORTIE
}

void loop()
{
  digitalWrite(ledPin, HIGH);   // allume la LED
}
```

digitalRead(broche)

Lit l''état (= le niveau logique) d'une broche précise en entrée numérique, et renvoie la valeur HIGH (HAUT en anglais) ou LOW (BAS en anglais).

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

Entrées analogiques

- **analogRead(broche_analogique)**

Lit la valeur de la tension présente sur la broche spécifiée.

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

Sorties "analogiques" (génération d'impulsion)

- ***analogWrite(broche, valeur);***

Génère une impulsion de largeur / période voulue sur une broche de la carte Arduino (onde PWM - Pulse Width Modulation en anglais ou MLI - Modulation de Largeur d'Impulsion en français). Ceci peut-être utilisé pour faire briller une LED avec une luminosité variable ou contrôler un moteur à des vitesses variables.

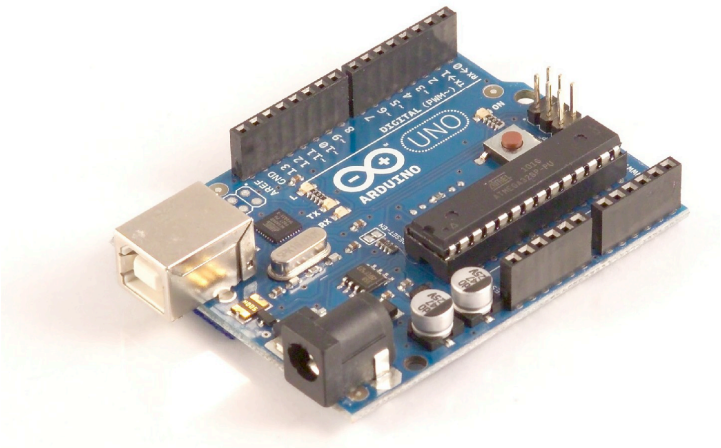
broche: la broche utilisée pour "écrire" l'impulsion.
valeur: la largeur du "duty cycle" (proportion de l'onde carrée qui est au niveau HAUT) : entre 0 (0% HAUT donc toujours au niveau BAS) et 255 (100% HAUT donc toujours au niveau HAUT).

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

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

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

Pour la suite aller faire un tour à l'adresse suivante :

http://arduino.cc/fr/Main/Reference

XI – La carte Arduino Uno

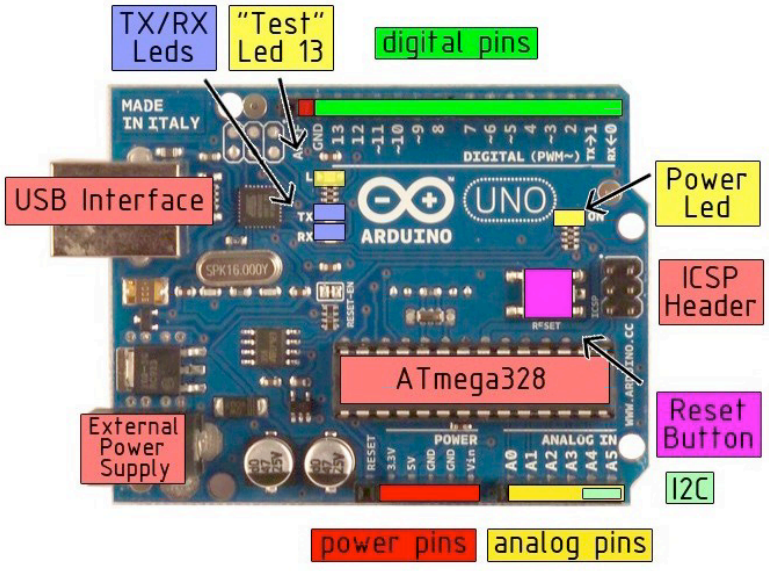


La carte Arduino uno

Caractéristiques de la carte Arduino uno

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

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



Les « shields »

Un« shield » Arduino est une petite carte qui se connecte sur une carte Arduino pour augmenter ses fonctionnalités. Quelques exemples de « shields » :

- Afficheur graphique
- Ethernet et carte SD
- GPS
- Carte de prototypage (type labdec)
- Etc…

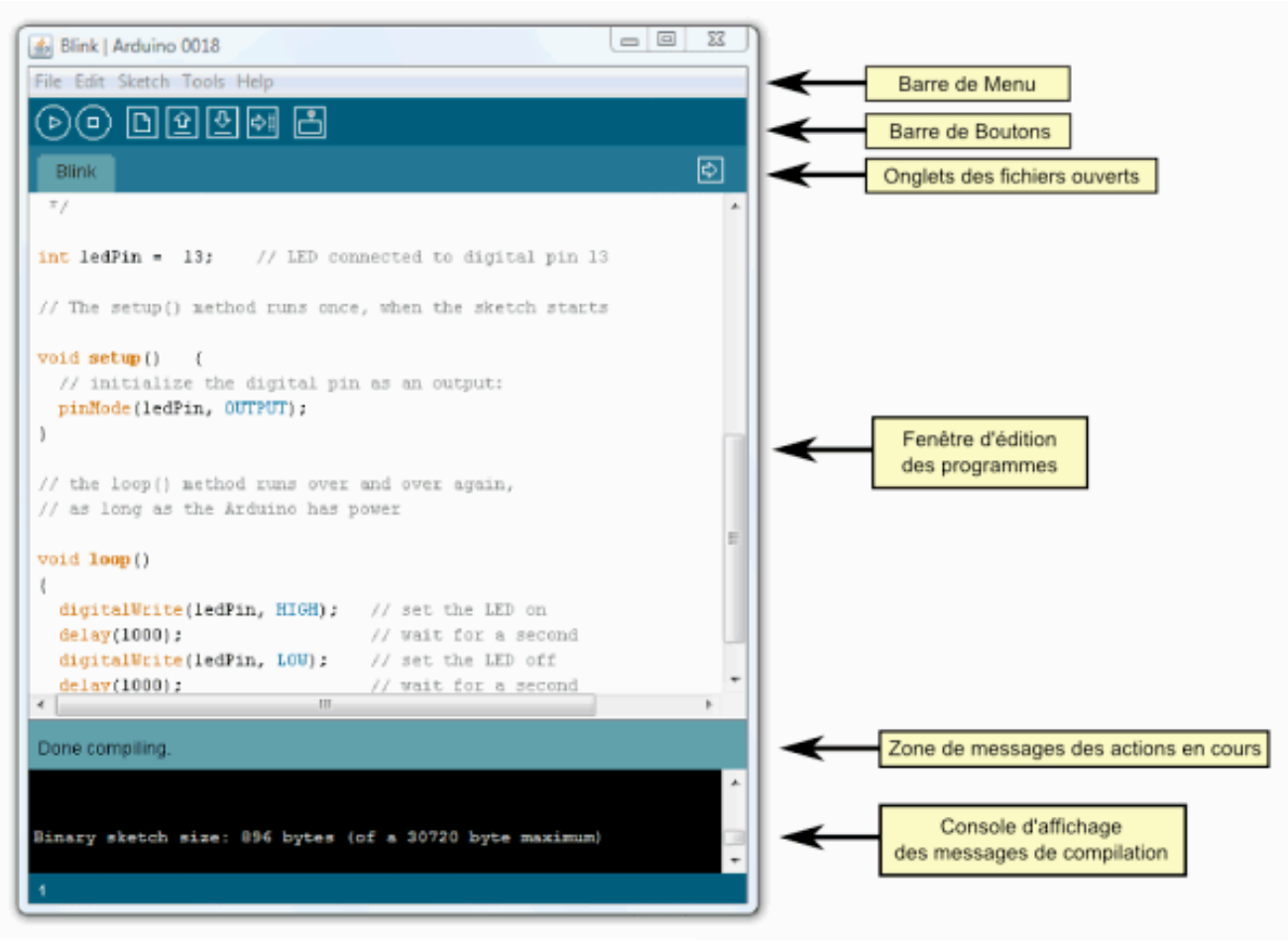
Arduino uno + shield Ethernet

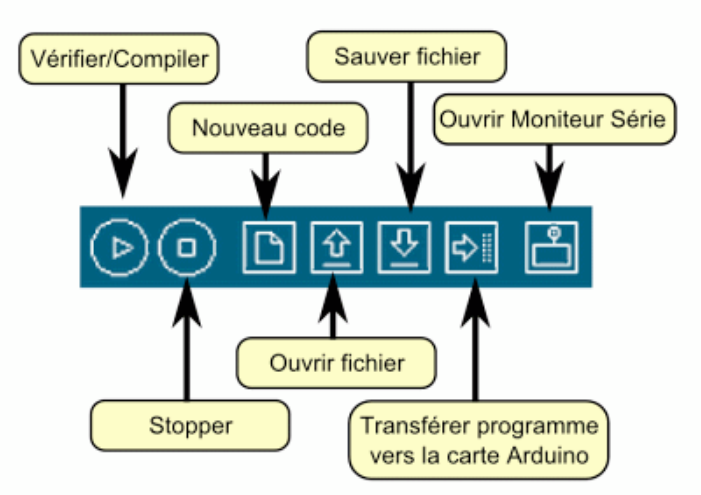
Développement d'un projet

Le développement sur Arduino est très simple :

- on code l'application : comme on l'a déjà vu, le langage Arduino est basé sur les langages C/C++ , avec des fonctions et des librairies spécifiques à Arduino (gestions des e/s).
- on relie la carte Arduino au PC et on transfert le programme sur la carte,
- on peut utiliser le circuit !

Le logiciel de programmation des modules Arduino est une application Java multiplateformes (fonctionnant sur tout système d'exploitation), servant d'éditeur de code et de compilateur, et qui peut transférer le firmware (et le programme) au travers de la liaison série (RS232, Bluetooth ou USB selon le module).





Détail de la barre de boutons

Le logiciel comprends aussi un moniteur série (équivalent à hyperterminal) qui permet de d'afficher des messages textes émis par la carte Arduino et d'envoyer des caractères vers la carte Arduino (en phase de fonctionnement).