1. Cartes Arduino

Les cartes Arduino sont conçues pour réaliser des prototypes et des maquettes de cartes électroniques pour l'informatique embarquée.

Ces cartes permettent 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).

Le langage Arduino se distingue des langages utilisés dans l'industrie de l'informatique embarquée par sa simplicité. En effet, beaucoup de librairies et de fonctionnalités de base occulte certains aspects de la programmation de logiciel embarquée afin de gagner en simplicité. Cela en fait un langage parfait pour réaliser des prototypes ou des petites applications dans le cadre de hobby.

2. Qu'est-ce qu'un microcontrôleur ?

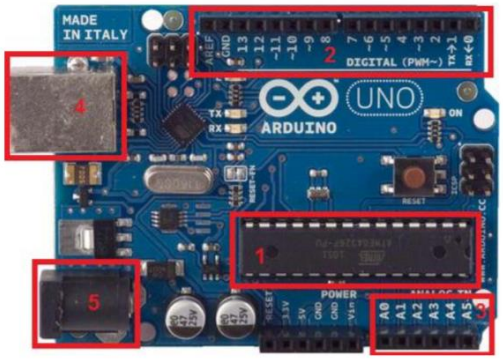
Les cartes Arduino font partie de la famille des microcontrôleurs.

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, …

On peut comparer un microcontrôleur à un ordinateur classique, mais sans système d'exploitation et avec une puissance de calcul considérablement plus faible. Les microcontrôleurs sont inévitables dans les domaines de l'informatique embarquée, de l'automatique et de l'informatique industrielle. Ils permettent de réduire le nombre de composant et de simplifier la création de cartes électroniques logiques.

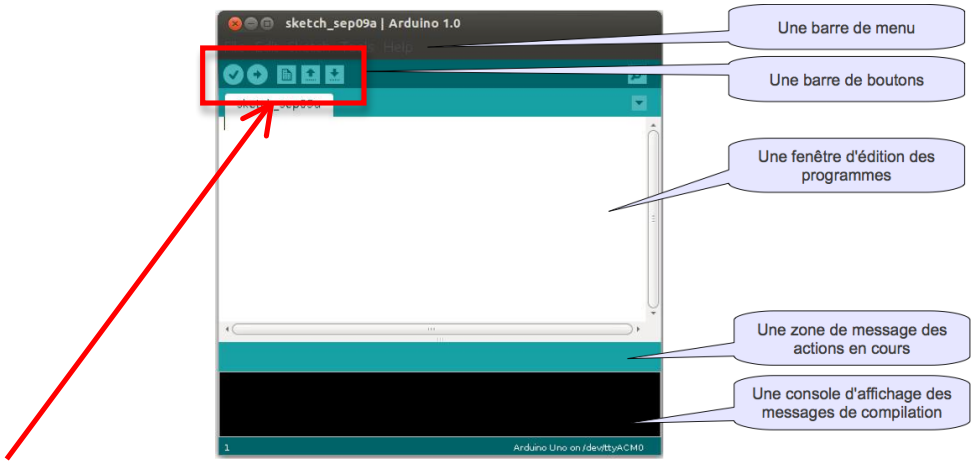
3. Arduino UNO



Composant	Fonction
1. Microprocesseur	Faire des calculs, stocker le programme
2. Connecteur (PIN) Entrées/Sorties digitales (0 ou 1)	Acquérir ou Commander
3. Connecteur (PIN) Entrées Analogiques	Acquérir
4. Connecteur USB	Alimenter la carte, transférer le programme
5. Connecteur alimentation	Alimenter la carte en 7-12V

4. IDE

Environnement de Développement Intégré (IDE) dédié au langage Arduino et à la programmation des cartes Arduino :



Le bouton "**Vérifier**", pour vérifier votre programme : Il faut en effet que le programme écrit ne présente pas de bugs afin de s'exécuter correctement.

Ensuite, le bouton "**Téléverser**" : En cliquant sur ce bouton, vous pouvez transférer votre programme compilé dans la mémoire de votre carte Arduino.

Au milieu, le bouton "**Nouveau**" : C'est à l'aide de ce bouton que vous pouvez créer de nouveaux programmes.

Après, le bouton "**Ouvrir**" (flèche vers le haut) qui vous permet d'accéder aux programmes d'exemples de l'IDE ou aux programmes présents sur votre machine.

Enfin, le bouton "**Enregistrer**" (flèche vers le bas) avec lequel vous pouvez sauvegarder le travail que vous avez réalisé afin d'y revenir quand vous le souhaitez.

5. Le langage Arduino

Le langage de programmation Arduino peut être divisé en trois parties principales : Variables, structures et fonctions.

Dans un programme en C/C++ :

- Une instruction se termine toujours par un ;
- Les { } servent à délimiter le début et la fin d'une instruction
- Un commentaire est un texte qui n'est pas lu par le compilateur. On y met généralement des informations explicatives pour le lecteur ou le programmeur. Pour écrire un commentaire, il faut mettre // avant chaque phrase. Ou alors, vous pouvez mettre /* au début et */ à la fin du commentaire entier.
- Il ne faut JAMAIS mettre d'accent ou d'espace dans les variables d'un programme

5.1. Les variables

Nom	Contenu	Taille (octet)	Plage de valeur	Préfixe variable	Exemple
char	Entier ou caractère	1	[-128 ; 127]	c	cLettre
int	Entier	2	[-32768 ; 32767]	i	iNombre
long	Entier	4	[-2 147 483 648 ; -2 147 483 647]	l	lDistance
float	Nombre à virgule	4	[-3,4028235E+38 ; 3,4028235E+38]	f	fTaille
string	Chaine de caractères	variable	Aucune	s	sPrenom
boolean	Booléen	1	[0 ; 1] = False / True	b	bInterrupteur

On utilisera aussi **#define** pour donner un nom à une valeur constante. Comme pour les numéros de PIN de l'arduino

Exemple : #define PINLED 12

Arduino utilise aussi des constantes pour simplifier le paramétrage de fonction :

Constante	Description
HIGH	Niveau logique haut
LOW	Niveau logique bas
INPUT	Entrée
OUTPUT	Sortie
TRUE	Vrai
FALSE	Faux

5.2. Structures de contrôle et opérateurs

Les structures de contrôle sont des blocs d'instructions qui s'exécutent en fonction du respect de condition ou de test. Le résultat d'un test est booléen (Vrai ou Faux).

Pour effectuer un test il faut utiliser des opérateurs.

- == (égal à)
- != (différent de)
- < (inférieur à)
- > (supérieur à)
- <= (inférieur ou égal à)
- >= (supérieur ou égal à)

Nom	Utilité	Syntaxe

if	Condition logique	if (**test**) {
		  <instruction>
		}

if else	Condition logique	if (**test**) {
		  <instruction>
		}
		else {
		  <instruction>
		}

if, else if	Condition logique multiple	if (**test1**) {
		  <instruction>
		}
		else if (**test2**) {
		  <instruction>
		}

for	Boucle itérative	for (<initialisation> ; <**test**>; <évolution>) {
		  <instruction>
		}

do while	Boucle	do { <instruction> } while (**test**)
switch case	Sélecteur	switch (<variable>) { case <valeur> : <instruction> break ; default : <instruction> }

5.3. Les fonctions

Le langage Arduino vient avec un nombre important de **fonction de base** permettant d'interagir avec son **environnement**. Les fonctions les plus utilisées sont les fonctions d'entrées/sorties. Ce sont elles qui permettent **d'envoyer ou de mesurer** une tension sur une des broches de la carte. Mais il existe une infinité d'autres fonctions plus ou moins complexes.

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

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.

Dans un premier temps, avant d'effectuer une mesure ou d'envoyer une commande. Il est nécessaire de définir la direction des broches utilisées. Pour cela on fait appel à la fonction **pinMode** (à utiliser dans le void **setup()**)en lui donnant d'une part, la broche concernée, et d'autre part, la direction :

```
void setup() {
    pinMode(1,OUTPUT) ;          // Définition broche 1 en sortie
    pinMode(2,INPUT) ;           // Définition broche 2 en entrée
}
```

5.3.1. Fonction de base à utiliser dans le void loop()

- **digitalRead(pin)** : Mesure une donnée numérique sur une des broches, la broche en question doit être réglée en entrée.
- **digitalWrite(pin, value)** : Écrit une donnée numérique sur une des broches, la broche concernée doit être réglée en sortie. Le paramètre value doit être égal à HIGH (état 1 soit 5V) ou LOW (état 0 soit 0V).
- **analogRead(pin)** : Mesure une donnée analogique sur une des broches (compatible seulement), la broche doit être réglée sur entrée.
- **analogWrite(pin, value)** : Écrit une donnée sous forme de PWM sur une des broches (compatible uniquement), la broche doit être réglée en sortie. Le paramètre value doit être compris dans l'intervalle [0;255]
- **delay(value)** : Permet de faire une temporisation en milliseconde de durée value

6. Exemples

6.1. Code avec un delay

```
#define PINLED 13                       // PINLED prend la valeur 13

void setup() {
    pinMode(PINLED, OUTPUT);            // Broche 13 en sortie
}

void loop() {
    delay(500);                         // Attente d'une demi seconde
    digitalWrite(pinLed, HIGH);         // Allumage de la LED
    delay(500);                         // Attente d'une demi seconde
    digitalWrite(pinLed, LOW);          // Extinction de la LED
}
```

6.2. Code avec un if

```
#define PINLED 13                       // PINLED prend la valeur 13
#define PINBP 12                        // PINBP prend la valeur 12, BP pour Bouton Poussoir

bool bEtat = 0 ;                        // Création de la variable bEtat

void setup() {
    pinMode(PINLED, OUTPUT);            // Broche 13 en sortie
    pinMode(PINBP, INPUT);              // Broche 12 en entrée
}

void loop() {
    bEtat = digitalRead(PINBP);
    if ( bEtat == 1){                   // Test du Bouton Poussoir, égal 1 si enfoncé
    digitalWrite(PINLED, HIGH);         // Allumage de la LED
    }
    else{                               //Bouton Poussoir pas enfoncé
    digitalWrite(PINLED, LOW);          // Extinction de la LED
    }
}
```