



Atelier Arduino

Références

<https://www.arduino.cc/reference/fr/>

http://arduino.education/wp-content/uploads/2018/01/Arduino_cours.pdf

https://www-soc.lip6.fr/trac/sesi-peri/chrome/site/docs/Aide-memoire_Arduino.pdf

Sommaire

- 0- Présentation de l'atelier et du radio club**
- 1- Notions d'électricité et d'électronique**
- 2- Installation et appropriation de VSC**
- 3- Faire clignoter une LED, étude du programme**
- 4- Le microcontrôleur**
- 5- Système décimal, binaire, hexadécimal – Code ASCII**
- 6- Ecrire un programme qui lance un SOS – les fonctions**
- 7- Organisation de la mémoire – les variables**
- 8- Les opérateurs – les conditions et structures**
- 9- L'Organigramme ou l'algorithme**
- 10- Les sorties PWM**
- 11- Les entrées numériques**
- 12- Les entrées analogiques**
- 13- Etude et réalisation des différents projets**

1- Notions d'électricité et d'électronique

Arduino_cours.pdf page : 18

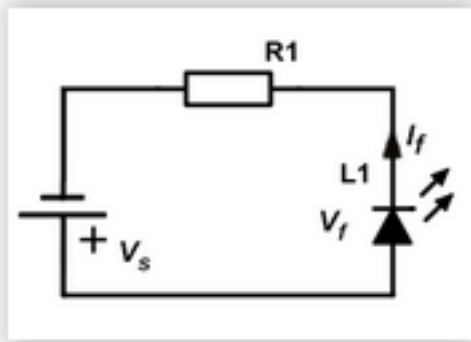
Tension - Intensité - Puissance - Résistance - LED – Potentiomètre
Transistor

Exercices : Calculer la résistance R1

Luminosité faible = 5mA

Luminosité moyenne = 8 à 10mA

Luminosité forte = 15 à 20mA

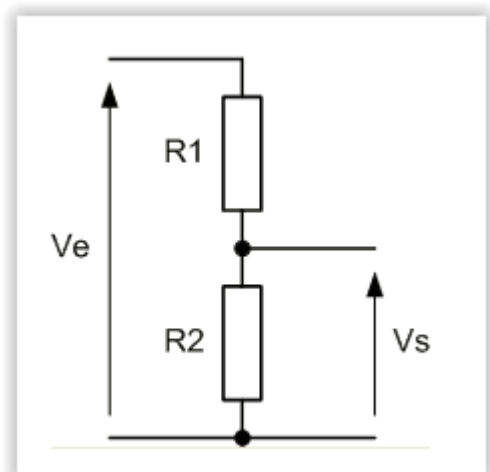


Couleur de LED	Tension admissible (Volt)
Rouge	1,8 à 2,1
Ambre	2 à 2,2
Orange	1,9 à 2,2
Jaune	1,9 à 2,2
Vert	2 à 3,1
Bleu	3 à 3,7
Blanc	3 à 3,4

Pont diviseur de tension

Le **pont diviseur** de tension est un montage électronique simple qui permet de **diviser une tension d'entrée**. Pour réaliser un pont diviseur de tension, il suffit de connecter deux résistances en série.

$$V_s = V_e \times \frac{R_2}{R_1 + R_2}$$



Exercices : **Calculer la valeur de la tension V_s**

$$V_e = 5\text{v} \quad R_1 = 1\text{k} \quad R_2 = 2,2\text{k}$$

Calculer la valeur de R_1 et R_2

$$V_e = 5\text{v} \quad V_s = 3\text{v} \quad I = 10\text{mA}$$

Calculer la puissance dissipée par les résistances

2- Installation et appropriation de VSC

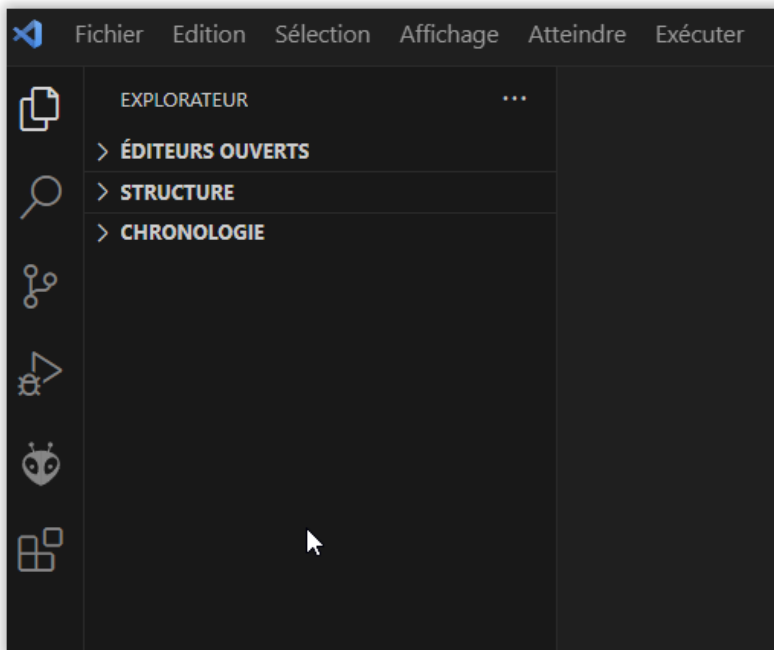
- Vidéo d'installation IDE Arduino et VSCode

<https://www.youtube.com/watch?v=8PKhW8kA-8E>

- Installation de VSC :

<https://code.visualstudio.com/download>

- Paramétrage de VSC :



Module linguistique français pour VSC

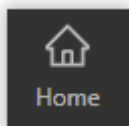
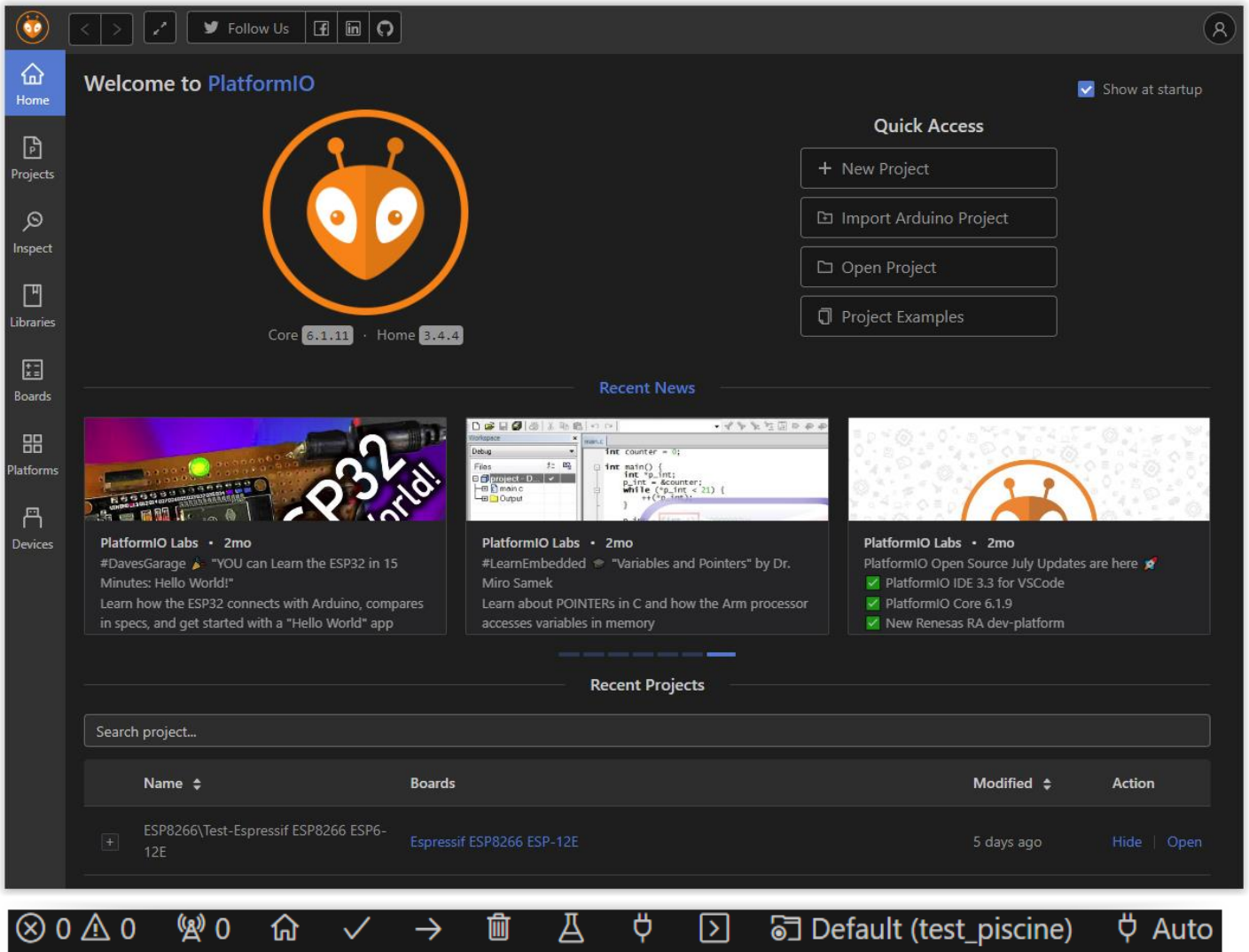


Extention PlatformIO pour Arduino

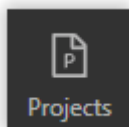


Live serveurur

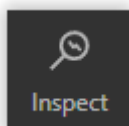
➤ Présentation de PlatformIO



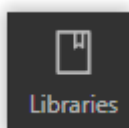
Affiche la page de "**Bienvenue**" de PlatformIO



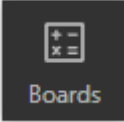
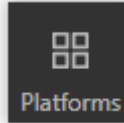
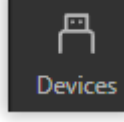











Affiche les projets sous PlatformIO



Contrôle les ressources utilisées par votre projet

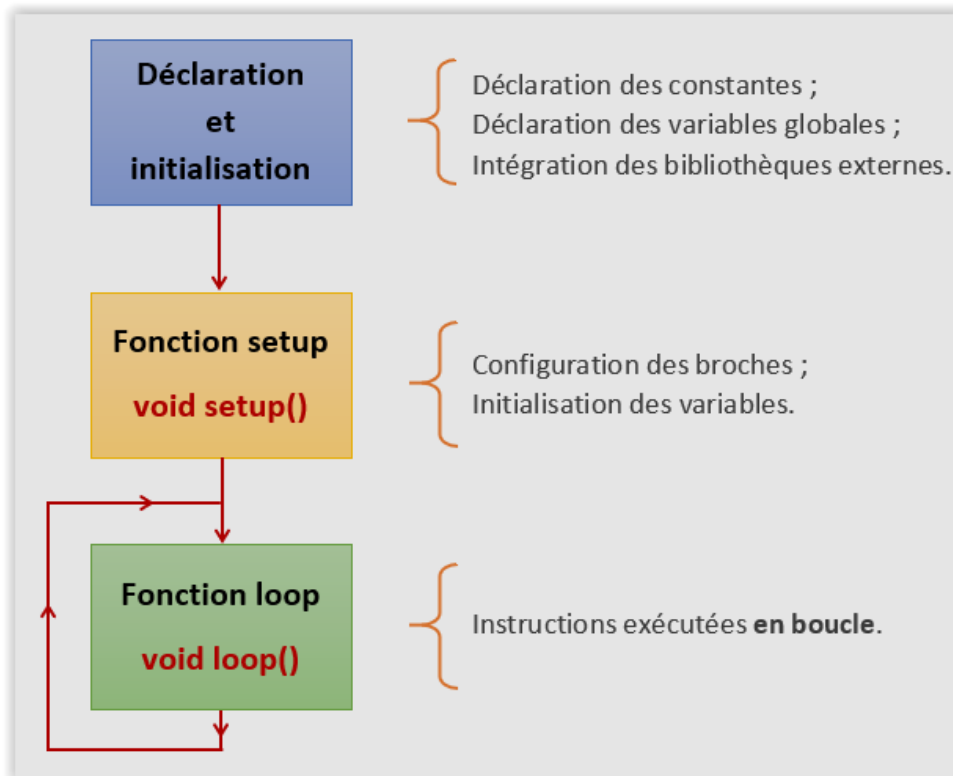


Affiche les librairies

-  Affiche cartes reconnues par PlatformIO
-  Gère les cartes installées
-  Affiche le port Com utilisé
-  0 Affiche le nombre d'erreur dans l'écriture du code
-  0 Affiche le nombre de tâche en fonctionnement
-  Affiche la page de "**Bienvenue**" de PlatformIO
-  Exécute la vérification et la construction du code
-  Téléverse le code dans la carte
-  Efface le terminal
-  Teste le code
-  Affiche le moniteur série
-  Ouvre un nouveau terminal
-  **Default (test_piscine)** Affiche le projet en cours
-  **Auto** Détection du port automatique

➤ Premier programme avec VSC

Squelette d'un programme Arduino



Programme minimum

```
#include <Arduino.h>

// Déclaration des variables (optionnelle)

void setup() {

  // Initialisation et configuration des entrées/sorties
}

void loop() {

  // la fonction loop() s'exécute en boucle aussi longtemps que l'Arduino est sous tension
}
```

3- Faire clignoter une LED, étude du programme

```
1  #include <Arduino.h>
2  // Faire clignoter la LED connectée sur la broche 13
3
4  //Déclaration des variables (optionnelle)
5  const byte brocheLED = 13; // Affecte la broche 13 à la constante brocheLed
6
7  //Initialisation et configuration des entrées/sorties
8
9  void setup() {
10     pinMode(brocheLED, OUTPUT); // configure la broche 13 en SORTIE
11     Serial.begin(9600);         // Fixe le débit de communication avec le port serie
12 }
13
14
15 // la fonction loop() s'exécute en boucle aussi longtemps que l'Arduino est sous tension
16
17 void loop() {
18     digitalWrite(brocheLED, HIGH); // Met le niveau logique haut (5v) sur la broche numérique
19     delay(100);                    // Réalise une pause dans l'exécution du programme pour la durée (en millisecondes)
20     digitalWrite(brocheLED, LOW);  // Met le niveau logique bas (0v) sur la broche numérique.
21     delay(100);                    // Réalise une pause dans l'exécution du programme pour la durée (en millisecondes)
22 }
```


#include <Arduino.h> Appel à la bibliothèque Arduino

Les bibliothèques (ou librairies) sont un ensemble de fonctions permettant de simplifier l'utilisation d'un capteur ou d'une fonctionnalité. Dès qu'un programme Arduino contient une ligne commençant par #include alors, il appelle une librairie.

Les commentaires // ou /* */

Les commentaires sont des lignes de code qui seront ignorées par le programme. Elles ne servent à rien lors de l'exécution du programme. Ils permettent d'annoter et de commenter le programme.

Les instructions

Les instructions sont des lignes de code qui disent au programme : « fais ceci, fais cela... » Ce sont donc les ordres qui seront exécutés par l'Arduino.

// Déclaration des variables (optionnelle)

const byte brocheLed = 13 ;

const byte	types de données
brocheLed	nom de la donnée
=	signe d'affectation
13	Valeur

Le ; Le point-virgule indique la fin d'une instruction

Les fonctions

Les fonctions sont une séquence d'instructions réalisant un calcul ou une tâche.

La 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 qu'une seule fois.

La fonction **loop()** (boucle en français) est exécutée des milliers de fois par seconde tant que la carte est sous tension.

Les parenthèses ()

Elles encadrent les paramètres passés à une fonction.
Elles permettent de définir l'ordre d'exécution des instructions

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.

void setup()

void	type de fonction
setup	nom de la fonction
([para1,para2,..])	paramètres de la fonction
{ instructions }	instructions

pinMode(brocheLed, OUTPUT);

pinMode	Instruction ou fonction
brocheLed	numéro de la broche
OUTPUT	Mode [OUTPUT INPUT]

void loop()

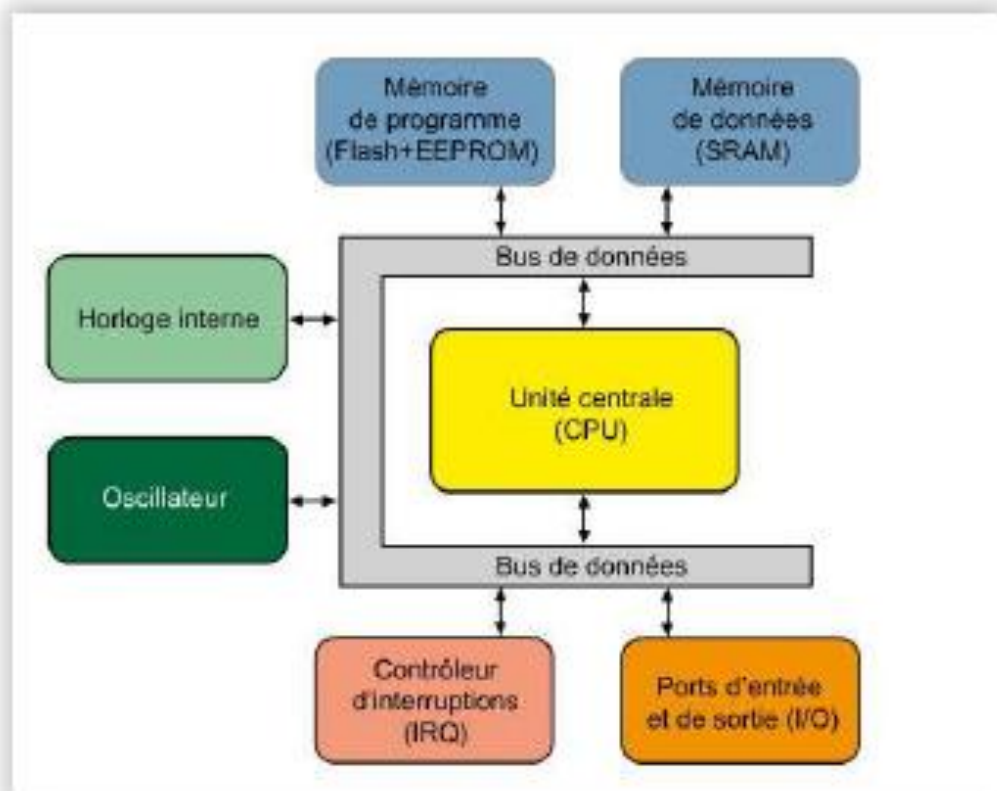
digitalWrite(brocheLed, HIGH);

digitalWrite	Instruction ou fonction
brocheLed	numéro de la broche (LED_BUILTIN LED interne)
HIGH	niveau [HIGH LOW]

delay(1000);

delay	Instruction ou fonction
1000	temps en ms

4- Le microcontrôleur



Microcontrôleur : C'est un circuit intégré qui intègre sur une puce plusieurs éléments complexes dans un espace réduit, (un véritable petit ordinateur).

CPU : Unité centrale de traitement (*Cerveau de l'homme*), une des pièces les plus importantes d'un ordinateur.

SRAM : La mémoire **SRAM** (Static Read Access Memory), sert à stocker des données temporaires (les variables du programme). C'est une mémoire volatile.

Flash La mémoire **FLASH** sert à stocker les programmes à exécuter, c'est une mémoire qui perdure après arrêt de l'alimentation.

EEPROM La mémoire **EEPROM** : (Electrically-Erasable Programmable Read-Only Memory) sert à stocker des données persistantes du programme.

Bus de données Permet d'échanger des informations entre les différents composants

Horloge interne Permet de piloter l'unité centrale

Oscillateur Permet de cadencer les informations sur la liaison série.

Cont. d'inter. Permet de gérer les interruptions

Ports E/S Permet les connexions entre le microcontrôleur et le monde extérieur

5- Système décimal, binaire, hexadécimal – Code ASCII

Décimal	Binaire	Hexadécimal
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Qu'est-ce que le code ASCII

La mémoire de l'ordinateur conserve toutes les données sous forme numérique. Il n'existe pas de méthode pour stocker directement les caractères. Chaque caractère possède donc son équivalent en code numérique : c'est le **code ASCII** (*American Standard Code for Information Interchange*)

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	\$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[59	3B	;	91	5B	[123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-`	63	3F	?	95	5F	`	127	7F	DEL

6- Ecrire un programme qui lance un SOS

➤ Cahier des charges :

Un SOS correspond à 3 signaux courts suivi de 3 longs puis à nouveau de 3 signaux courts



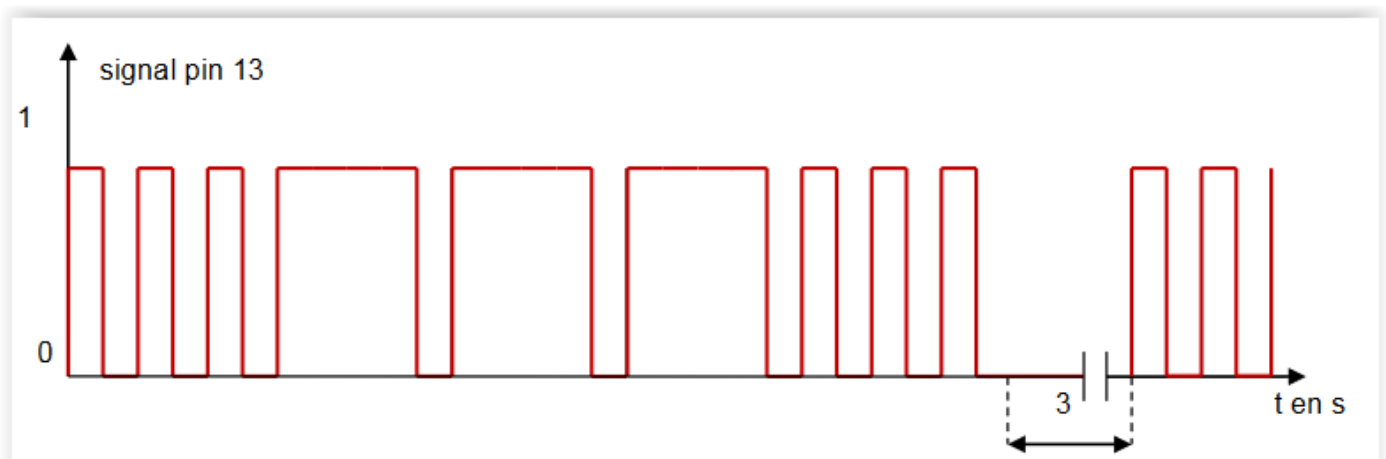
Point = 150ms

Trait = 400ms

Espacement entre deux signaux = 150ms

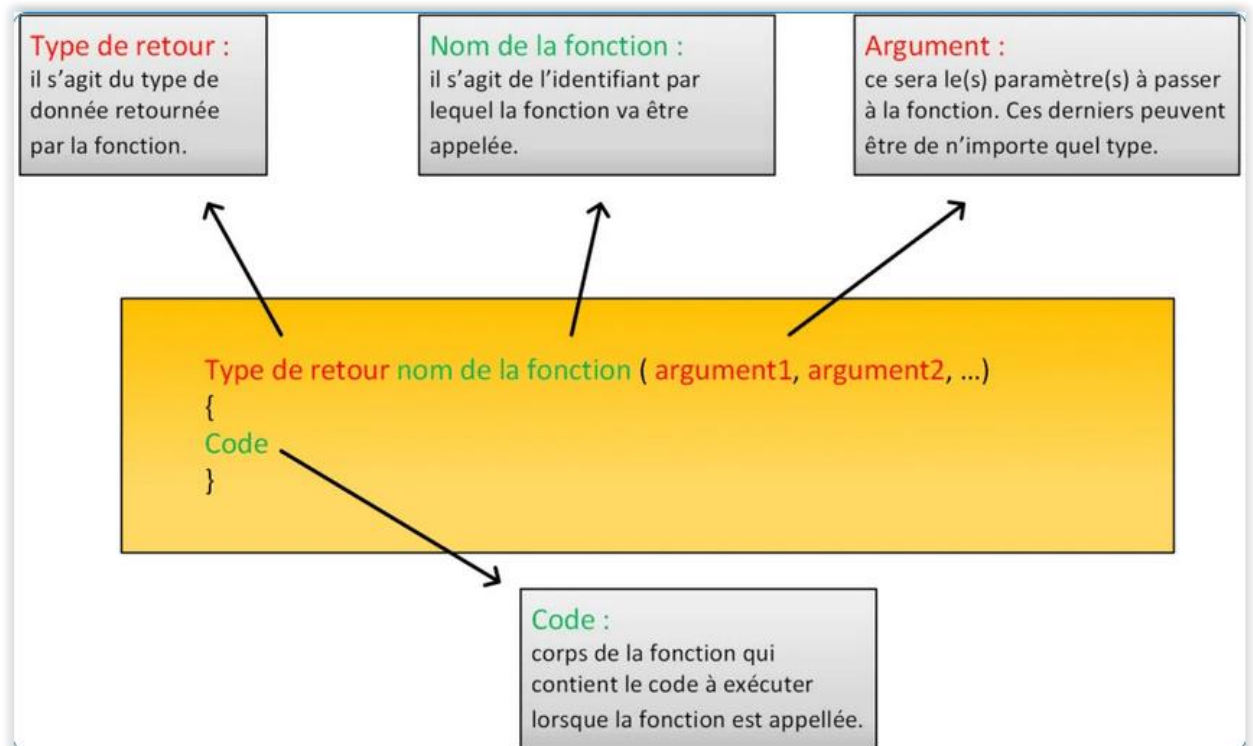
Espacement entre deux SOS = 1s

➤ Chronogramme du signal



➤ Les fonctions

Une **fonction** est une séquence d'instructions réalisant un calcul ou une tâche. Une fonction peut posséder des paramètres d'entrée (des **arguments**) et peut également retourner des valeurs de sortie



Ex :

```
int somme(int a, int b)
{
    int c = 0 ;
    c = a + b ;
    return c ;
}

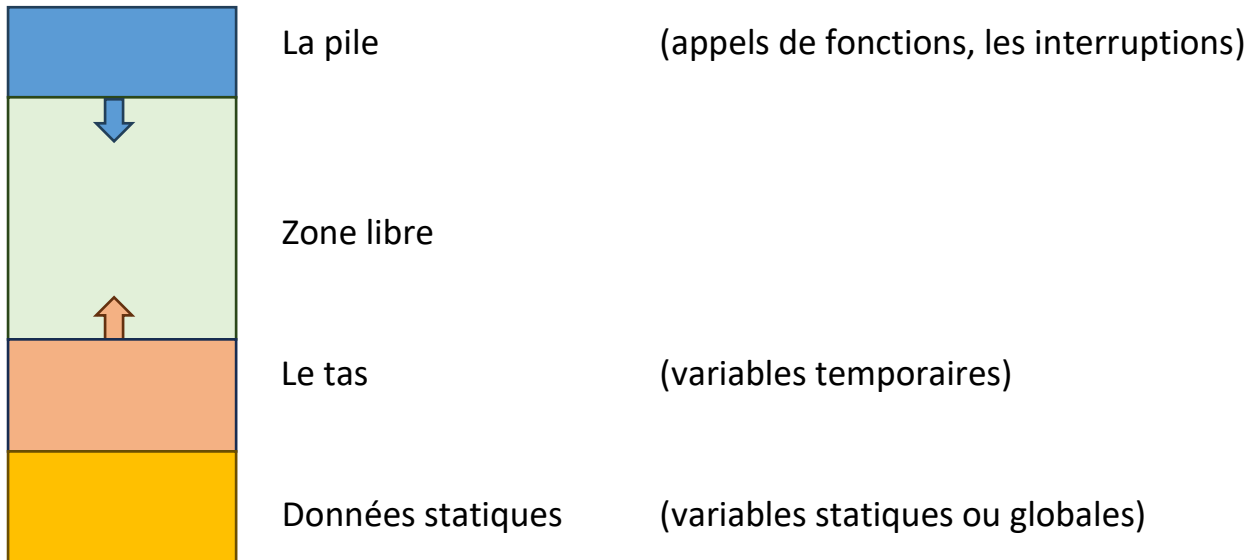
void setup()
{
    Serial.begin(9600) ;
    int resultat = somme (6, 4) ;
    Serial.print("Somme de a + b = ") ;
    Serial.print(resultat) ;
}

void loop() {}
```

7- Organisation de la mémoire – les variables

➤ Utilisation de la SRAM :

Pour stocker des données d'usages différents, l'espace mémoire est découpé en 3 zones



➤ Les variables :

Définir leur type, c'est à dire le genre de données qu'elle contiendra (nombre entier, à virgule, ou caractère) et sa taille.

Ex : `int compt = 0 ;`

Déclarer une constante :

Ex : `const int8_t led_rouge = 13;`

`#define led_rouge = 13`

Avec le mot-clé **const**, est la manière la plus recommandée pour déclarer une constante dans un programme.

Avec le **#define**, la constante sera ici prise en charge par le préprocesseur.

Déclarer une variable :

Les entiers

Données codées sur 8 bits

- **char** ou **int8_t** : valeurs de -128 à 127
- **unsigned char** ou **uint8_t** : valeurs de 0 à 255
- **byte** : valeurs de 0 à 255

Données codées sur 16 bits

- **int** ou **int16_t** : valeurs de -32768 à +2767
- **unsigned int** ou **uint16_t** : valeurs de 0 à 65535

Données codées sur 32 bits

- **long** ou **int32_t** : valeurs de -2^{31} à $2^{31} - 1$
- **unsigned long** ou **uint32_t** : valeurs de 0 à $2^{32} - 1$

Les réels

Simple précision (32 bits) :

- **float** : valeurs de $\pm 3,4 \cdot 10^{-38}$ à $\pm 3,4 \cdot 10^{+38}$

Double précision (64 bits) :

- **double** : valeurs de $\pm 1,7 \cdot 10^{-308}$ à $\pm 1,7 \cdot 10^{+308}$

Les caractères

- **char** : caractère codé avec un entier de 0 à 127

Ex : **'A' = 65** **'a' = 97**

Forme générale

- type nom [= valeur] ;
- type nom1 [= valeur], Nom[X] [= valeur] ;

Ex : **int compt = 0;**

int i = 3, j = 50;

float f = 2.05

char ch1 = 'a', ch2 = 'Z';

Portée d'une variable

On appelle portée d'une variable l'ensemble des endroits du programme où elle existe.

Globales :

Déclarées en dehors de toutes fonctions ou classes.
Visible dans tout le programme.

Locales :

Déclarées dans une fonction ou un bloc
Visible uniquement dans la fonction ou le bloc

Ex:

```
long duree = 3;
```

```
void setup () {  
  int l;  
}
```

```
void loop () {  
  int i = 25;          static int i = 25;   garde sa valeur initiale  
  float x = 0.1;  
  
  i = i*2;  
  duree = 3*duree;  
}
```

8- Les opérateurs – les conditions et structures

Les opérateurs

Arithmétiques

Opérateur d'affectation : =

Addition +

Soustraction -

Multiplication *

Division /

Modulo %

Relationnels

Egalité : ==

Différent de : !=

Supérieur : >

Inférieur : <

Supérieur ou égal : >=

Inférieur ou égal : <=

Logiques

ET : &&

OU : ||

NON : !

Résultat : vrai ou faux

Condition renvoie 1 si elle est vraie, 0 si elle est fausse

Les conditions et structures

if / else

```
if (condition) {  
    actions ;  
}
```

```
if (condition) {  
    actions ;  
}  
else {  
    actions ;  
}
```

```
if (condition) {  
    actions ;  
}  
else if (condition) {  
    actions ;  
}  
else {  
    actions ;  
}
```

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

switch / case

```
switch(expression)
{
    case valeur 1 :
        action si expression est égale à valeur 1
        break ;

    case valeur 2 :
        action si expression est égale à valeur 2
        break ;

    case valeur 3 :
        action si expression est égale à valeur 3
        break ;

    default :
        action si expression est égale à valeur 1
        break ;
}
```

```
switch (var)
{
    case 1:
        // faire l'action A
        break;
    case 2:
        // faire l'action B
        break;
    case 3:
        // faire l'action C
        break;

    default:
        // si aucune condition n'est vraie
        // faire l'action défaut
}
```

Boucle while

```
while(condition) // tant que la condition est vraie
{
    // instructions à exécuter
}

var = 0;
while(var < 20)
{
    var = var + 1
}
```

Boucle do while

```
do // faire...
{
    // instructions à exécuter
} while (condition); // tant que la condition est vraie

var = 0;
do
{
    var++
}
while (var < 20);
```

Boucle for

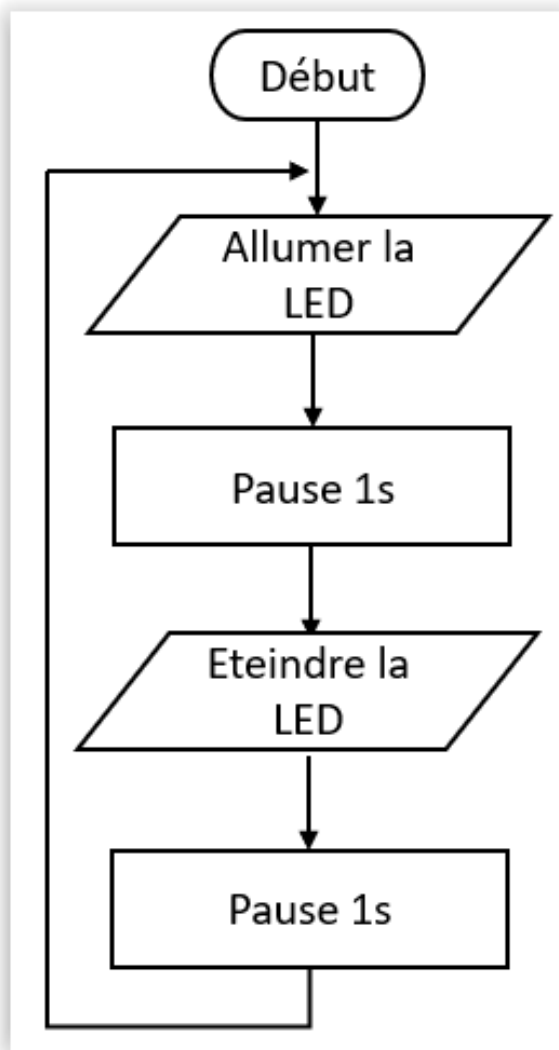
```
for (initialization; condition; incrementation) {
    // instructions à exécuter;
}

for (var = 0 ; var < 20 ; var++)
{
    Serial.print(".");
}
```

9- L'Organigramme ou l'algorithme

C'est la représentation graphique d'une suite structurée d'instructions (recette de cuisine)

Exemple



Symboles normalisés



Début, fin, interruption
Début, fin ou interruption d'un programme.



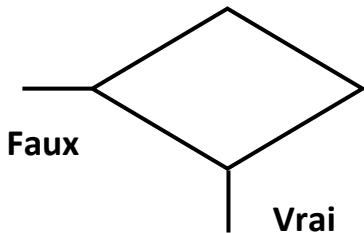
Traitement interne
Opération ou calcul sur des données dont le résultat est stocké dans le microcontrôleur.



Sous-programme
Portion de programme considérée comme une simple opération



Lecture ou écriture de données externes
Mise à disposition d'une information sur un port de sortie ou enregistrement d'une information d'un port d'entrée.



Branchement
Test, exploitation de conditions de variables
Symbole utilisé pour représenter une décision.

Programme

Un **programme** informatique est un ensemble d'instructions et d'opérations destinées à être exécutées par un ordinateur.

Les instructions

Les **instructions** sont des lignes de code qui disent au programme : "fais ceci, fais cela, ..."

La fonction

Une **fonction** est un ensemble d'instructions réalisant une certaine tâche.

10- Les sorties PWM

Tout d'abord **PWM** sont les Initiales de **Pulse Width Modulation** que l'on traduit en français par **MLI (Modulation de Largeur d'Impulsion)**.

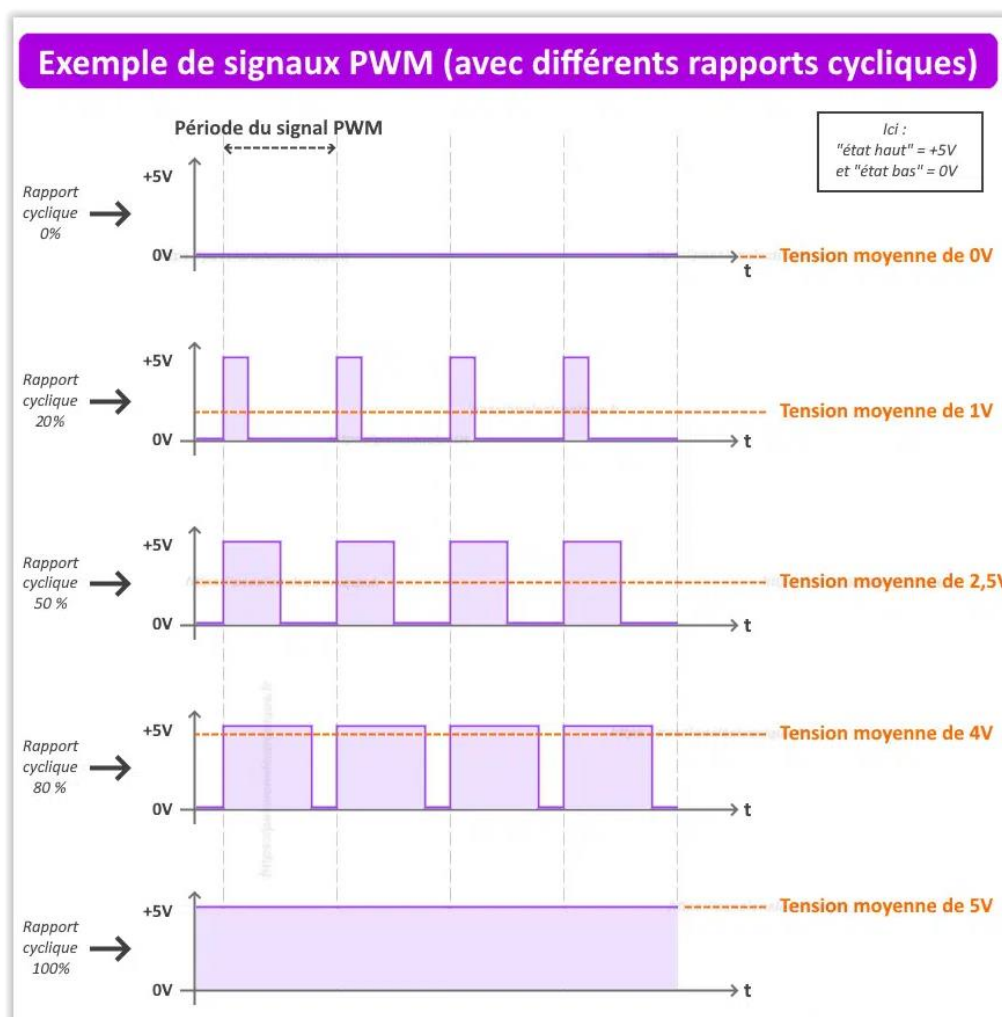
Quand on souhaite faire varier la **luminosité d'une diode (LED)**, la couleur des LEDs RGB ou faire de la variation de vitesse d'un moteur DC, il est nécessaire de générer un signal analogique.

L'Arduino est équipé de **broches analogiques** en entrées mais qui ne peuvent pas être utilisées en sorties.

Comment modifier la tension avec le signal PWM :

Pour **générer** une **tension variable** ou pseudo analogique en sortie d'une broche digitale de l'Arduino, il va falloir changer très rapidement l'état de la sortie. En effet, le fait de passer d'un état LOW à HIGH très vite va engendrer la variation de la valeur moyenne de la tension.

Or, c'est ce changement de la **valeur moyenne** du signal électrique qui va **changer** la luminosité d'une LED ou **varier** la vitesse d'un moteur.



Les sorties PWM sur un Arduino UNO sont : **D3, D5, D6, D9, D10, D11**

Elles sont repérées avec le symbole « ~ », placé juste devant le numéro de broche pouvant émettre un signal PWM.

Instruction :

```
analogWrite(numéro_broche,Valeur_PWM);
```

Valeur_PWM : comprise entre 0 et 255

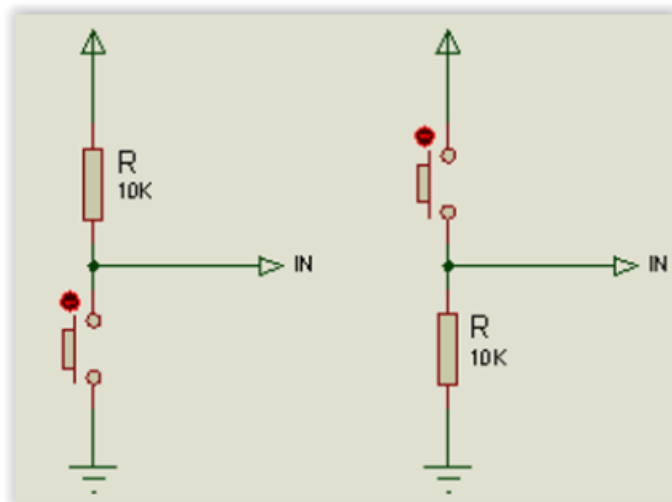
11- Les entrées numériques

Les entrées numériques sont utilisées pour lire l'état d'un interrupteur, d'un bouton poussoir ou recevoir une information en provenance d'un capteur tout ou rien.

Une tension supérieure à 3V sera vue comme à l'état HAUT, (= HIGH = true = 1)
Une tension inférieure à 0.5V et la sortie sera vue à l'état BAS (= LOW = false = 0)

➔ entre 0.5v et 3v c'est aléatoire, parfois 0 parfois 1

Pour éviter les tensions intermédiaires, on ajoute une résistance de rappel



Résistance de pull-up, sert à "tirer" ("to pull" in english) le potentiel vers le haut (up).

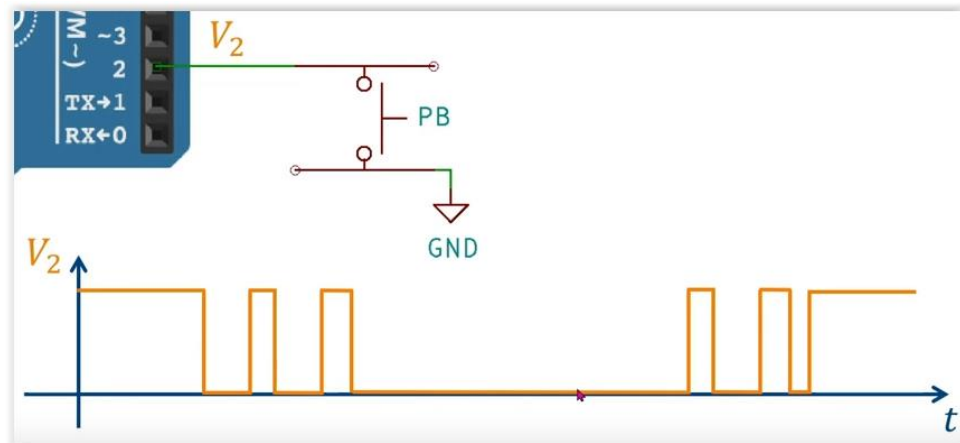
Résistance de pull-down sert à "tirer" le potentiel vers le bas (down)

Instruction :

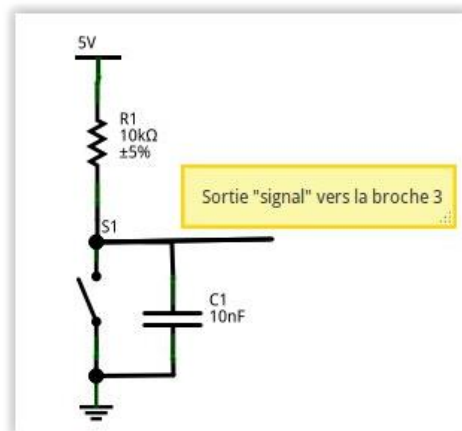
```
pinMode (numéro_broche, [INPUT] | [INPUT_PULLUP]);
```

```
digital.Read(numéro_broche);
```

Les boutons ne sont pas des systèmes mécaniques parfaits. Du coup, lorsqu'un appui est fait dessus, le signal ne passe pas immédiatement et proprement de 5V à 0V. En l'espace de quelques millisecondes, le signal va "sauter" entre 5V et 0V plusieurs fois avant de se stabiliser. Il se passe le même phénomène lorsque l'utilisateur relâche le bouton.



Solution matérielle :



Solution logicielle :

Utilisation des fonctions **millis()** ou **micros()** que nous verrons plus tard.

Utilisation de la bibliothèque **ezButton.h**

Exercice :

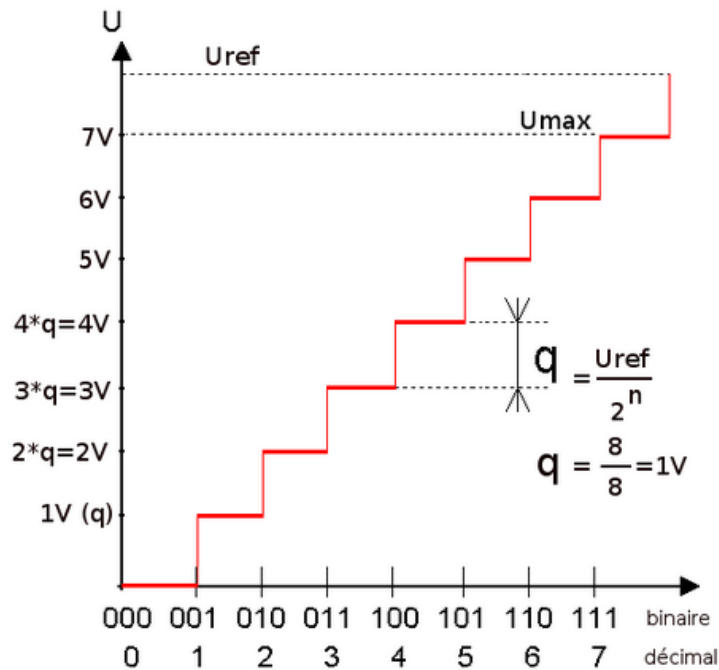
Déclaration et utilisation des bibliothèques

12- Les entrées analogiques

C'est un dispositif qui va convertir des grandeurs analogiques en grandeurs numériques. La valeur numérique obtenue sera proportionnelle à la valeur analogique fournie en entrée.

Ce dispositif s'appelle un convertisseur analogique numérique (CAN)

Exemple d'un convertisseur analogique→numérique (CAN) alimenté en 8V (U_{ref}) sur 3 bits:



Remarques:

- ✚ Pour retrouver la valeur numérique fournie par un CAN, il suffira de diviser la tension d'entrée par la résolution. Ainsi si $U=4V$ alors le nombre numérique correspondant sera $4/1=4$ (100 en binaire), si $U=5$ on aura $5/1=5$ (101 en binaire), ...

$$N = \frac{V_e}{q}$$

Les entrées analogiques sur un Arduino UNO sont notées A0 à A5.

Nous pouvons appliquer sur ces entrées des tensions comprises entre 0 et 5V.

Le convertisseur analogique-numérique transforme ces tensions en mots binaires de 10 bits.

L'instruction permettant de lancer la conversion analogique-numérique est :

```
analog.Read(numéro_broche);
```

La relation qui permet de calculer la tension présente à l'entrée est :

$$V_e = (V_{ref} / 1024) CDA$$

Utilisation de la fonction `map()` (*Faire une mise à l'échelle*)

Cette fonction transforme une valeur, initialement dans un intervalle déterminé, en une autre valeur comprise dans un nouvel intervalle de façon que cette modification soit la plus proportionnelle possible.

```
map(valeur_entrée, inférieur_entrée, supérieur_entrée,  
inférieur_sortie, supérieur_sortie);
```

valeur_entrée -> c'est le nombre à mapper (à transformer)

inférieur_entrée -> c'est la limite inférieure de l'intervalle actuel

supérieur_entrée -> c'est la limite supérieure de l'intervalle actuel

inférieur_sortie -> c'est la limite inférieure de l'intervalle résultat du changement

supérieur_sortie -> c'est la limite supérieure de l'intervalle résultat du changement

```
ex : map(value, 0, 1023, 0, 255);
```

Exercices :

Ex10 : Mesurer la tension présente à l'entrée d'une broche Arduino « en l'air ».

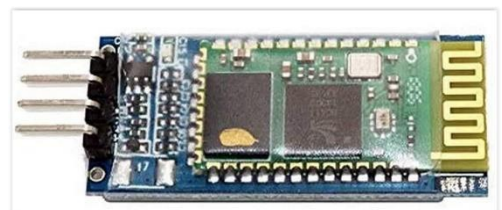
Ex11 : Lecture d'une entrée analogique commandée par un potentiomètre.

Ex12 : Commander la vitesse d'un moteur à l'aide d'un potentiomètre.

13 - Etude et réalisation des différents projets

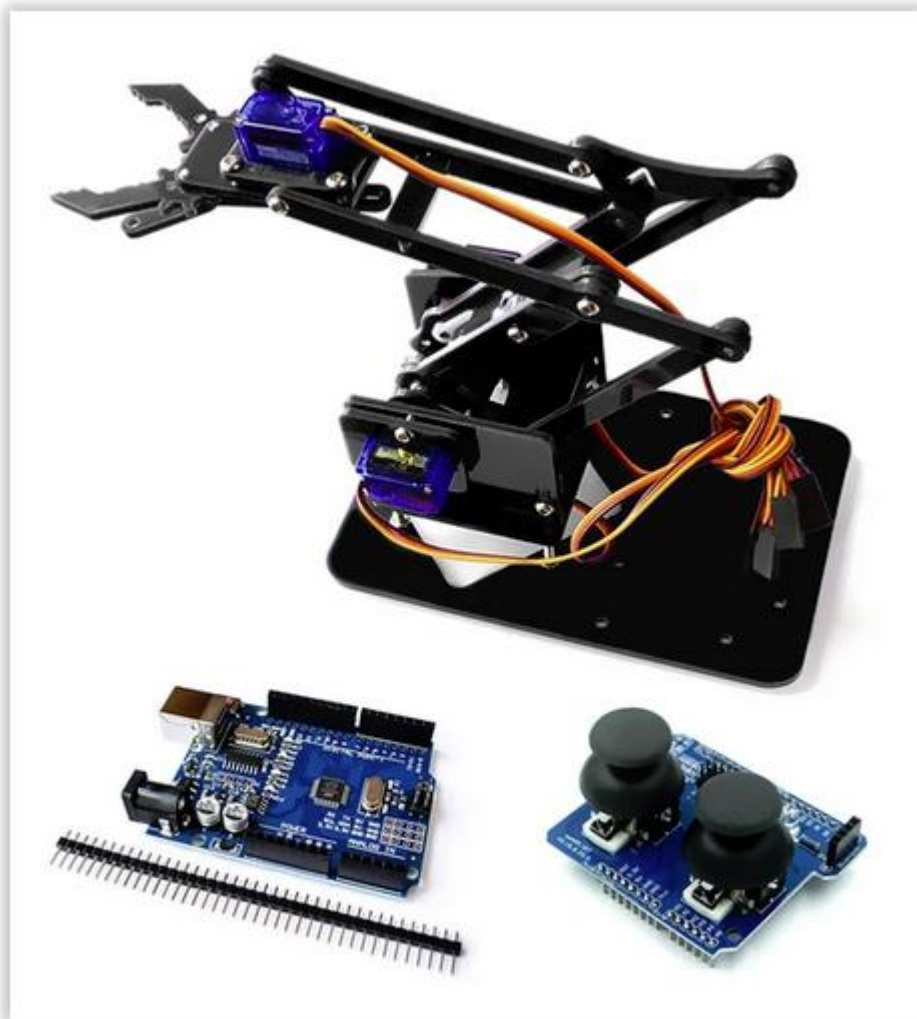
Voiture robot : matériels et modules utilisés

- Kit de châssis de voiture Robot intelligent
- Module Bluetooth HC-06
- Entretoises hexagonales, vis et écrous
- Câbles de raccordement



Bras robotique : matériels et modules utilisés

- Un ensemble de pièces de coupe en acrylique
- Un paquet de vis
- 4 servomoteurs sg90
- 1 Arduino uno
- 1 module joystick double PS2



Montage :

https://www.bilibili.com/video/BV1Ed4y1c7sL/?spm=a2g0s.imconversation.0.0.107f3e5flnhR1U&spm_id_from=333.337.search-card.all.click&vd_source=cc670f71805a95b1bf5b2d7278ca76ca

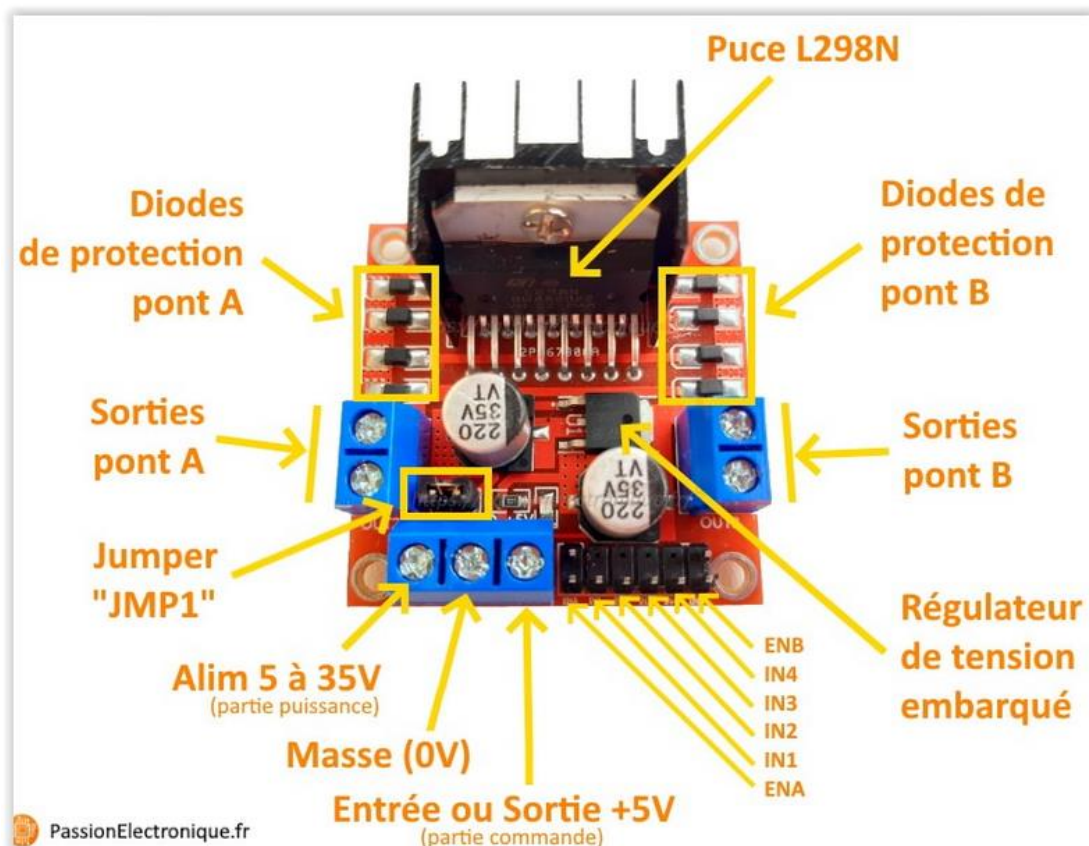
Le moteur à courant continu



Les moteurs à courant continu possèdent souvent une boîte de réduction afin d'augmenter leur couple pour un encombrement réduit.

Le moteur CC est très simple d'utilisation. Pour le faire fonctionner, il suffit d'appliquer une tension électrique à ses bornes. Le signe et le niveau de cette tension vont imposer le sens et la vitesse de rotation.

Carte module L298



Avec le cavalier régulateur on peut contrôler des moteurs de 5 à 12 V et on pourra utiliser la broche de 5 V du module pour alimenter l'Arduino.

Sans ce cavalier régulateur, on peut contrôler des moteurs de 12 à 35 V, et dans ce cas la troisième broche agira comme entrée de 5 V pour alimenter la partie logique du driver.


Les sorties des moteurs A et B nous fourniront l'énergie nécessaire pour démarrer les moteurs. Assurez-vous que **la polarité** des moteurs est la même sur les deux entrées. Dans le cas contraire, vous devrez peut-être les échanger lorsque vous réglerez les deux moteurs vers l'avant et vers l'arrière.

Les pins IN 1 et IN 2 nous servent à contrôler le sens de rotation du moteur A, et les pins IN 3 et IN 4, celui du moteur B.

Table de vérité L298N - Fonctionnement

Entrées de sélection				
ENA	IN1	IN2	Résultat, en sortie	1er pont (A) 2ème pont (B)
ENB	IN3	IN4	Résultat, en sortie	
L	X	X	Moteur en roue libre (à l'arrêt, sans frein)	
H	L	L	Blocage du moteur (arrêt rapide, freinage fort)	
	L	H	Marche arrière	
	H	L	Marche avant	
	H	H	Blocage du moteur (arrêt rapide, freinage fort)	

X = peu importe
L = état bas (0V, par exemple)
H = état haut (+5V, par exemple)

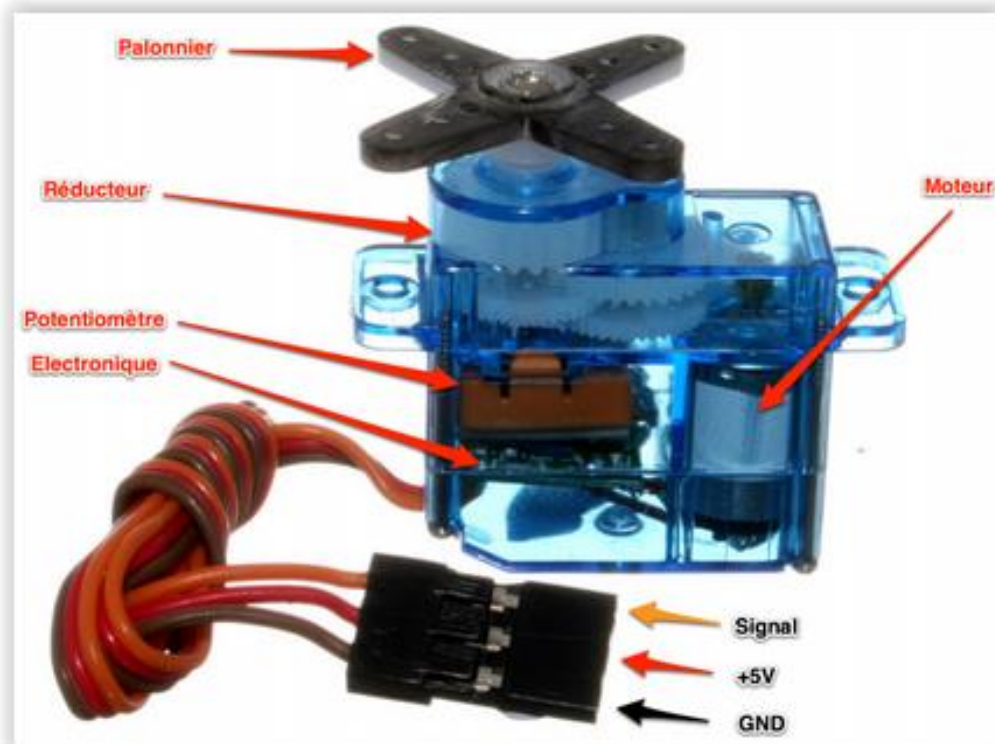
 PassionElectronique.

Pour contrôler la vitesse de rotation des moteurs on doit retirer les cavaliers des pins ENA et ENB.

On les connecte à deux sorties PWM de la plaque Arduino afin qu'on puisse envoyer une valeur entre 0 et 255 pour gérer cette vitesse des moteurs.

Avec les cavaliers installés, les moteurs tourneront toujours à la même vitesse.

Etude de servomoteur



Les servomoteurs sont contrôlés en envoyant un signal PWM (modulation de largeur d'impulsion) à l'entrée signal du servo.

La largeur des impulsions détermine la position de l'arbre de sortie.

Lorsque vous envoyez au servo un signal avec une largeur d'impulsion de 1,5 millisecondes (ms), le servo se déplace vers la position neutre (90 degrés).

Les positions min (0 degré) et max (180 degrés) correspondent respectivement à une largeur d'impulsion de 1 ms et 2 ms.

Pour piloter un servomoteur, nous allons utiliser la librairie **Servo.h**

Pour connecter notre servomoteur à la broche de l'Arduino, nous utiliserons la méthode **attach(n° broche)** de la librairie.

attach(pin, min, max)

pin: le numéro de la broche à laquelle le servo est attaché.

min: (facultatif): la largeur d'impulsion, en microsecondes, correspondant à l'angle minimal (0 degré) sur le servo (valeur par défaut de 544).

max: (facultatif): la largeur d'impulsion, en microsecondes, correspondant à l'angle maximal (180 degrés) sur le servo (valeur par défaut de 2400).

write(angle en degré)

La méthode `write(...)` permet de donner l'angle auquel le servo doit se positionner.

Exemple : positionnez l'axe du servomoteur à 90°

```
#include <Servo.h>           // On commence par inclure la bibliothèque
Servo monServo;             // On crée un objet de type "Servo", nommé -> monServo

void setup()
{
  monServo.attach(2); // On attache le servo à la broche 2
  monServo.write(90); // On place le servo à 90°
}

void loop(){}
```

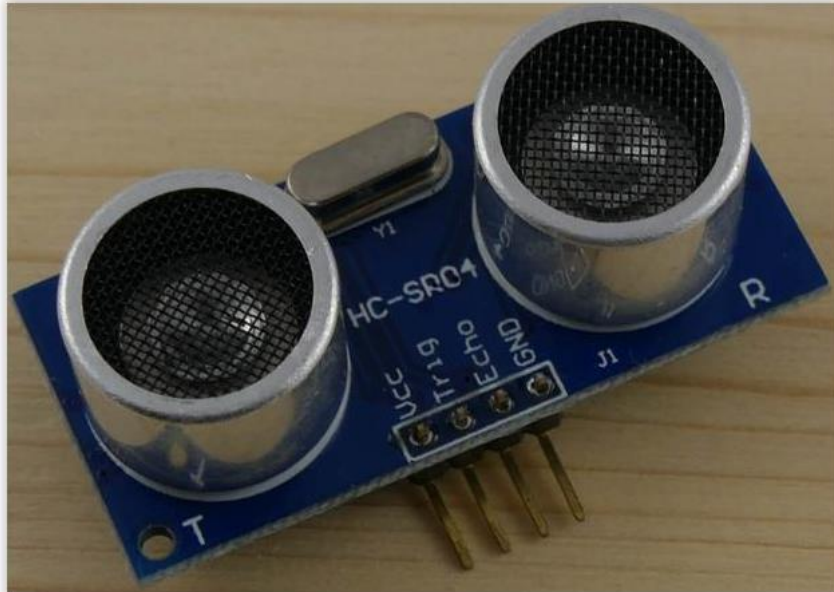
Exercice EX13 :

Faire varier la position du bras du servo de 0° à 180°
avec la méthode `write()`

Exercice EX14 :

Faire varier la position du bras du servo de 0° à 180°
avec la méthode `writeMicroseconds()`

Capteur à ultrason HC-SR04

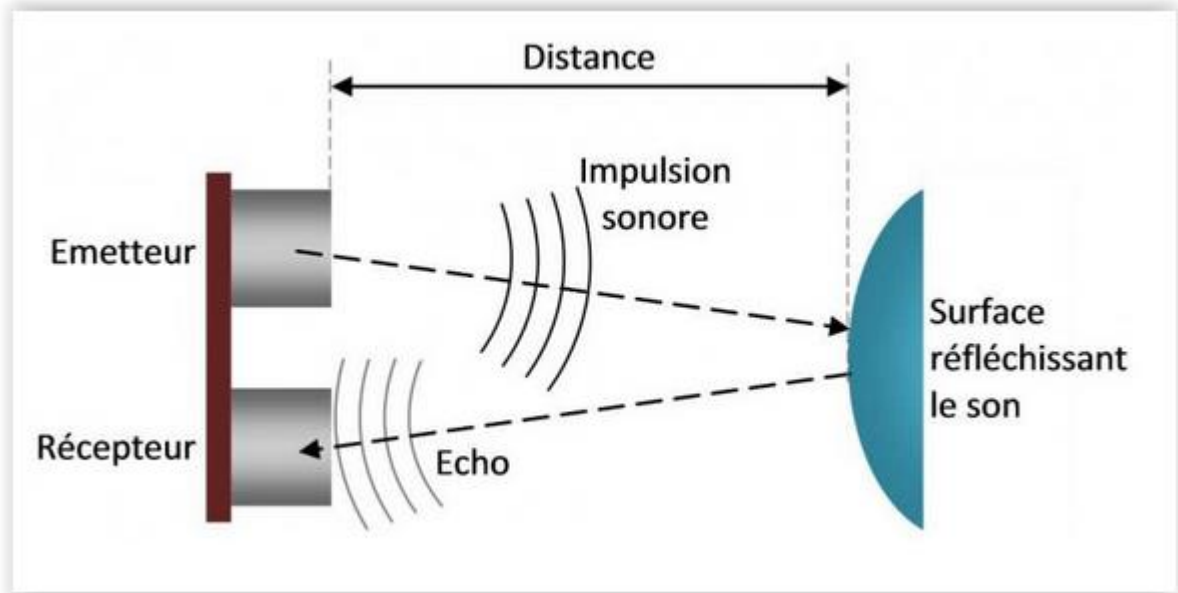


Ce capteur fonctionne avec une tension d'alimentation de 5 volts, dispose d'un angle de mesure de 15° environ et permet de faire des mesures de distance entre 2 centimètres et 4 mètres avec une précision de 3mm (en théorie, dans la pratique ce n'est pas tout à fait exact).

Le principe de fonctionnement du capteur est entièrement basé sur la vitesse du son.

Voilà comment se déroule une prise de mesure :

1. On envoie une impulsion HIGH de 10 μ s sur la broche TRIGGER du capteur.
2. Le capteur envoie alors une série de 8 impulsions ultrasoniques à 40KHz (inaudible pour l'être humain).
3. Les ultrasons se propagent dans l'air jusqu'à toucher un obstacle et retournent dans l'autre sens vers le capteur.
4. Le capteur détecte l'écho et clôture la prise de mesure.

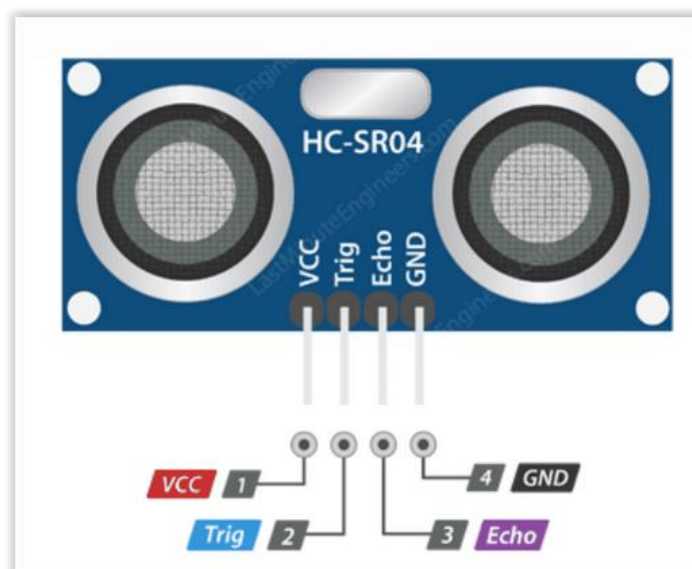


La **durée** entre l'instant de l'émission et l'instant de la réception peut être mesurée. Le signal ayant parcouru 2 fois la **distance** entre le capteur et la surface (un aller-retour), on peut la calculer ainsi :

$$\text{Distance} = \text{vitesse du son} \times \text{durée} / 2$$

(Vitesse du son est environ égale à 340 m/s.)

Pour simplifier les calculs et l'accès aux données nous allons utiliser la bibliothèque « Ultrasonic.h »



Exercice : Prendre l'exemple joint à la librairie.
A vous de jouer

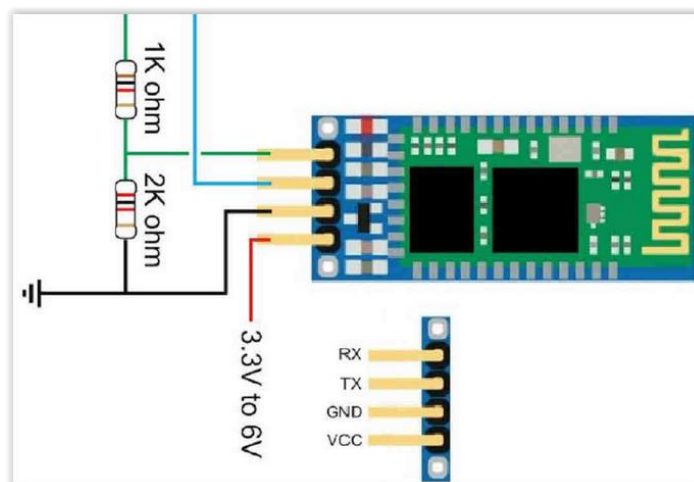
Module Bluetooth HC-06



Le module Bluetooth HC-06 **permet d'établir une liaison Bluetooth (liaison série) entre une carte Arduino et un autre équipement possédant une connexion Bluetooth** (Smartphone, tablette, seconde carte Arduino, etc....).



La tension d'alimentation de ces modules doit être comprise entre 3,3 et 5 V, mais la broche RX ne peut recevoir qu'une tension maximale de 3,3 V. Il faudra prévoir un pont diviseur de tension pour ramener la tension de 5 V délivrée par la carte Arduino à 3,3 V pour ne pas endommager la broche RX du module Bluetooth



Il faudra bien penser à inverser les broches Rx et Tx entre la carte Arduino et le module Bluetooth

