FICHES ARDUINO

CONCEPTS de FONCTION et de CLASSE

Une instruction Arduino est une fonction, sa fiche technique va donc indiquer :

* Son rôle, ce qu’elle fait ( = description )
* Sa manière de l’écrire ( = syntaxe )
* Le ou les paramètres qu’elle reçoit
* La valeur renvoyée

On trouve les fiches techniques de toutes les fonctions Arduino ( 30 ) depuis le logiciel Arduino – Menu « Help » 🡺 « Référence »

**On appelle classe un regroupement de fonctions**

Classe **Serial** rassemble toutes les fonctions utilisées pour la communication série USB

1°- *La fonction Serial.begin() 🡺 pour appeler la fonction begin() fonction*

*d’initialisation de la communication USB*

**Description :**

Fixe le débit de communication en nb de caractères par seconde ( l’unité est le baud pour la communication série.

**Syntaxe :**

Serial.begin( débit ) ;

**Paramètres :**

Débit : débit de communication en caractères par seconde ( 115200 ) fonctionne très bien.

**Valeur renvoyée :**

Aucune

Exemple : **Serial**.begin(115200) ;

2°- *Les fonctions Serial.print() et Serial.println 🡺 pour utiliser les fonctions*

*print() fonction d’affichage d’un message sans saut de ligne*

*println() fonction d’affichage avec saut de ligne*

**Description :**

Serial.println() : Affiche les données sur le port série suivi d’un passage à la ligne suivante.

Serial.print() : Affiche les données sur le port série sans passage à la ligne suivante.

**Syntaxe :**

Serial.println(data) ;

Serial.print(data) ;

**Paramètres:**

**Data**: tout type de données entières incluant les char ( chaînes de caractères ) et floats ( nombres à virgule ).

Les floats sont supportés avec une précision de 2 à plusieurs décimales.

**Valeur renvoyée :**

Aucun Exemple : **Serial**.println(« Salut ! ») ;

EXEMPLE : Envoi d’un message vers le PC via le port USB «  HELLO WORLD ! »

void **setup**() {

**Serial**.begin(112500) ;

}

void **loop**() {

**Serial**.println(« HELLO WORLD ! ») ;

delay(1000) ; // pause de 1 seconde

}

CONCEPT de VARIABLE

Lorsque l’on écrit un programme on va manipuler des nb entiers, des nb à virgule, des chaines de caractères. Pour pouvoir manipuler facilement toutes ces grandeurs on va les mémoriser, les stocker dans des « boites » avec des « étiquettes » dessus.

Une boite mémoire est une variable

* Dans la carte Arduino on dispose de plus de 1000 « boites » ( octets ) dans les quelles on va pouvoir lire, écrire, effacer des valeurs.
* Cette réserve de boites est la mémoire vive ( RAM )
* C’est elle que le programme Arduino utilise pour stoker les variables

En programmation on utilise des boites mémoire différentes en fonction de la « taille » ou de la nature de ce que l’on va mettre dedans.

Le type d’une variable c’est sa catégorie, son genre.

Le langage Arduino permet :

* De **créer des variables de types différents** en fonction des besoins
* De **les nommer**
* D’y **mettre des valeurs voulues**

*1°- Principe général de déclaration d’une variable*

Pour déclarer une variable il faut au minimum :

* Indiquer son **type** ( = ce que va contenir la « boite » )
* Indiquer son **nom** ( = l’étiquette à coller sur la « boite » )
* Suivi d’un signe **=** ce qui revient à ouvrir la « boite » pour mettre qqchose dedans
* Suivi de la **valeur initiale voulue** ( = ce que l’on met dans la « boite » )
* Finir la ligne par un **point-virgule ;** ( ne pas oublier !!!! )

*2°- Principaux types de variables disponibles dans le langage Arduino*

**Exemples de déclarations de variables**

a-) pour stoker une valeur pouvant prendre la valeur « 1 ou 0 » ou bien « vrai ou faux » on utilisera boolean

boolean maVariable=false ;

b-) pour stocker une valeur entière pouvant prendre comme valeur max 1000 on utilisera int

int maVariable=0 ;

c-) pour stoker une valeur entière pouvant prendre comme valeur max 1 000 000 on utilisera un long

long maVariable=50000 ;

d-) pour stoker une valeur décimale on utilisera un float

float maVariable=0.0 ;

Pour attribuer une valeur à une variale on fera :

maVariable=2500 ;

En pratique déclarer les variable en type int

*3°- Où déclarer une variable. Portée des variables*

Il est possible de déclarer une variable :

* **Soit avant** la fonction setup() dans une partie que l’on pourra appeler « entête déclarative »…la variable est accessible de n’importe où dans le programme, elle est dite «  globale »
* **Soit au sein d’une fonction** que ce soit la fonction setup() ou loop(). La variable n’existe qu’à l’intérieur de cette fonction, elle est dite « locale »
* **Soit au sein de certaine boucles** en tant que variables d’incrémentation

La **portée d’une variable** correspond à son « périmètre de visibilité » dans un programme.

*4°- Les variables fixes : les constantes*

Pour tous les types de variable, il existe une situation particulière : celle ou le contenu de la variable ne peut et ne doit pas changer au cours du programme :

* Si la variable représente le n° d’une broche
* Si la variable représente le nb π, etc……

On précisera que le contenu de la variable ne changera pas au cours du programme en créant une constante

On ajoute le mot clé const devant le type de la variable qui devient ainsi une constante.

Exemple : const int maBroche=13

Une fois déclarée la valeur de la constante ne peut plus être modifiée.

Plusieurs constantes prédéfinies dans le langage Arduino :

* HIGH et LOW correspondant à des niveaux de broches numériques
* INPUT et OUTPUT correspondant à des sens des broches numériques
* true et false correspondant aux états logiques VRAI et FAUX

En pratique utiliser une constante pour renommer une broche Arduino.

VARIABLES et FONCTIONS

**1°- Passer une variable en tant que paramètre d’une fonction**

Il est possible d’utiliser les variables avec les fonctions et donc avec les instructions Arduino de la même façon qu’on le ferait avec des valeurs numériques brutes. On va pouvoir :

* passer une variable en tant que paramètre d’une fonction
* récupérer le résultat renvoyé par une fonction dans une variable .

Il faut impérativement que la variable passée en paramètre soit **du même type que ce que la fonction de la définition prévoit :**

* par exemple la fonction delay() du langage Arduino reçoit un paramètre correspondant à la durée en millisecondes
* ce paramètre doit être de type int
* on pourra ainsi déclarer une variable int et passer à l’instruction delay()

Exemple : int duree=1000 ; // déclare une variable int valant 1000

delay(duree) ; // exécute l’instruction delay avec la valeur de la

variable

**2°- Récupérer le résultat renvoyé par une fonction dans une variable**

On pourra récupérer le résultat renvoyé par une fonction dans une variable du même type **simplement en posant l’égalité entre la variable et la fonction.**

* Par exemple la fonction millis() du langage Arduino renvoie une valeur correspondant au nb de millisecondes écoulées depuis le début du programme.
* La valeur renvoyée est de type long
* On pourra ainsi déclarer une variable long et récupérer le nb de millisecondes écoulées depuis le début du programme dans la variable simplement en posant l’égalité entre la variable et la fonction.

Exemple : long time=0 ; // déclare une variable time valant 0

time = millis() ; // le résultat renvoyé par la fonction est mis

dans la variable

STRUCTURE TYPE d’un LANGAGE ARDUINO UTILISANT des VARIABLES et des CONSTANTES

**Structure d’un programme Arduino type**

* Une **entête déclarative** avant les deux fonctions setup() et loop() au niveau de laquelle on déclarera :
* Les constantes dont la valeur ne changera pas et seront globales,
* Les variables globales qui seront accessibles de partout.
* La **fonction setup**() qui ne s’exécute qu’une seule fois dans laquelle on pourra :
* Egalement déclarer des variables locales si besoin,
* Initialiser si besoin les variables globales,
* Utiliser les constantes et les variables globales à volonté.
* Suivi de la **fonction loop**() qui s’exécute ensuite en boucle sans fin dans laquelle on pourra :
* Egalement déclarer des variables locales si besoin,
* Utiliser les constantes et les variables globales à volonté

**Afficher des variables dans le Terminal Série**

La structure est la suivante :

**Entête déclarative**

A ce niveau on va déclarer et initialiser

les variables utilisées, par exemple : //….entête déclarative

* Une variable int int myInt=-123 ;
* Une variable long long myLong=456789 ;
* Une variable float float myFloat=3,1416;
* Une constante int const int myConst=10 ;

**Fonction setup()** void **setup**() {

A ce niveau on va :

* Initialiser la communication série, **Serial**.begin(115200)
* +/- afficher les messages de

début de programme,

}

**Fonction loop()** void **loop**() {

A ce niveau on va :

* Afficher les messages utilisant **Serial**.print(‘’myInt=’’), **Serial**.println(myInt) ;

les variables, **Serial**.print(‘’myLong=’’), **Serial**.println(myLong) ;

* Faire une pause de 1 seconde **Serial**.print(‘’myFloat=’’), **Serial**.println(myFloat) ;

entre deux passages **Serial**.print(‘’myConst=’’), **Serial**.println(myConst) ;

**Serial**.println() ; //…saut de ligne vide

delay(1000) ; // pause de 1 seconde

LES INSTRUCTIONS du LANGAGE ARDUINO POUR la GESTION des BROCHES NUMERIQUES

Une broche numérique E /S est caractérisée par :

* Son sens : ENTREE ou SORTIE
* Son état : HAUT ou BAS

**Fixer le sens E /S d’une broche numérique :**

La première instruction à connaitre est l’instruction pinMode ( broche mode ) qui sert à fixer le sens ( ou mode de fonctionnement ) d’une broche numérique E / S avec :

* Broche : le n° de la broche de 0 à 19
* Mode : une des constantes prédéfinies suivantes :
* OUTPUT pour un fonctionnement en sortie
* INPUT pour un fonctionnement en entrée
* Cette fonction est à utiliser dans la fonction **setup**()

**Fixer le niveau HAUT/BAS d’une broche numérique :**

Si la broche est configurée en SORTIE on pourra contrôler son état ( ou « écrire » sur la broche ) à l’aide de la fonction digitalWrite ( broche, état ) avec :

* Broche : le n° de la broche de 0 à 19
* Valeur : une des constantes prédéfinies suivantes :
* HIGH : pour mettre la broche au niveau HAUT ( 5v ),
* LOW : pour mettre la broche au niveau BAS ( 0v ).

Si la broche est configurée en ENTREE, on pourra « lire » son état à l’aide de la fonction int digitalRead(broche) avec :

* Broche : le n° de broche de 0 à 19
* int : la valeur renvoyée par la fonction de type int

**Ecrire un programme qui met une broche en sortie au niveau HAUT :**

//….pas d’entête déclarative

void **setup**() { //…fonction setup() exécutée 1 fois en une seule fois

pinMode(13, OUTPUT) ; //…met la broche 13 en sortie

digitalWrite(13, HIGH) ; //…met la broche 13 au niveau HAUT

} //….fin de la fonction setup()

void **loop**() { //… fonction loop() exécutée ensuite en boucle sans fin

} //….fin de la fonction loop()

**Faire clignoter une LED : le programme**

//… pas d’entête déclarative

void **setup**() {

pinMode(2, OUTPUT) ; //…met la broche 2 en SORTIE

}

void **loop**() {

digitalWrite(2, HIGH) ; //… allume la LED

delay(1000) ; //…temps de pose 1 seconde

digitalWrite(2, LOW) ; //… éteint la LED

delay(1000) ; //…temps de pose de 1 seconde

}

**Même opération en utilisant une constante pour désigner la broche et une variable pour la vitesse de clignotement :**

const int LED=2

int vitesse=1000

void **setup**() {

pinMode(LED, OUTPUT) ;

}

void **loop**() {

digitalWrite(LED, HIGH) ;

delay(vitesse) ;

digitalWrite(LED, LOW) ;

delay(vitesse) ;

}

**En partique, pour écrire des programmes « propres » :**

* Renommer les broches dans l’entête déclarative avec des constantes !
* Pour toutes les valeurs qui reviennent souvent utiliser des variables !

**Faire clignoter 4 LEDs : le programme :**

const int LED1=2 ;

const int LED2=3;

const int LED3=4;

const int LED4=5;

int vitesse=1000;

void **setup**() {

pinMode(LED1, OUTPUT);

pinMode(LED2, OUTPUT);

pinMode(LED3, OUPUT);

pinMode(LED4, OUTPUT);

}

void **loop**() {

digitalWrite(LED1, HIGH);

digitalWrite(LED2, HIGH);

digitalWrite(LED3, HIGH);

digitalWrite(LED4, HIGH);

delay(vitesse);

digitalWrite(LED1, LOW);

digitalWrite(LED2, LOW);

digitalWrite(LED3, LOW);

digitalWrite(LED4, LOW);

delay(vitesse);

}

**Faire clignoter 8 LEDs en utilisant un tableau de constantes**

Quand le code risque de lourdement s’alourdir, on peut pour chaque variable simplement « répèter » le même code. Ceci sera rendu possible grâce à une boucle.

Mais pour parcourir une liste de variables à l’aide d’une boucle il faut au préalable créer un « tableau de constantes ».

const int LED[8] = {2,3,4,5,6,7,8,9} ;

int vitesse = 1000 ;

void **setup**() {

pinMode(LED0, OUTPUT) ;

pinMode(LED1, OUTPUT) ;

pinMode(LED2, OUTPUT);

pinMode(LED3, OUTPUT);

pinMode(LED4, OUTPUT);

pinMode(LED5, OUTPUT);

pinMode(LED6, OUTPUT);

pinMode(LED7, OUTPUT);

}

void **loop**() {

digitalWrite(LED0, HIGH);

digitalWrite(LED1, HIGH);

digitalWrite(LED2, HIGH);

digitalWrite(LED3, HIGH);

digitalWrite(LED4, HIGH);

digitalWrite(LED5, HIGH);

digitalWrite(LED6, HIGH);

digitalWrite(LED7, HIGH);

delay(vitesse);

digitalWrite(LED0, LOW);

digitalWrite(LED1, LOW);

digitalWrite(LED2, LOW);

digitalWrite(LED3, LOW);

digitalWrite(LED4, LOW);

digitalWrite(LED6, LOW);

digitalWrite(LED7, LOW);

delay(vitesse);

}

**On peut améliorer cette programmation: la boucle for :**

En programmation on rencontre fréquemment la situation où l’on doit répéter plusieurs fois **l’exécution d’un même code.**

Pour pouvoir faire cela simplement on utilise **une boucle** !

Il existe plusieurs types de boucles, la plus fréquente et la plus pratique est la boucle dite **for.**

La structure d’une boucle for s’apparente à celle d’une fonction avec cependant quelques ajouts :

* On commence par le mot clé for suivi de (
* Ensuite on déclare la **variable de comptage** et on fixe la valeur de départ. La variable étant locale, elle est limitée à la boucle…on fait suivre d’un ;
* Ensuite on fixe la valeur maximale à ne pas dépasser…on fait suivre à nouveau d’un ;
* Enfin on indique l’incrémentation à utiliser et on ferme la )
* Ensuite comme pour une fonction on met le code entre { }

//…entête déclarative

const int LED[8]= {2,3,4,5,6,7,8,9} ;

int vitesse = 1000 ;

void **setup**() {

for (int i=0 ; int i<8 ; i++) {

pinMode(LED[i], OUTPUT) ;

} //… fin de la boucle for

} //… fin de la fonction setup()

void **loop**() {

for (int i=0; int i<8; i++) {

digitalWrite(LED[i], HIGH);

} //… fin de for

delay(vitesse);

for (int i=0; int i<8; i++) {

digitalWrite(LED[i], LOW);

} //…fin de for

delay(vitesse);

} //…fin de la fonction loop()

**La condition if….else….**

Si l’on veut exécuter une instruction que dans certaines conditions ( *au lieu de faire* *clignoter toutes les LEDs, on en allume une puis les autres une à une de cette façon)*

L’instruction utilisée pour une **condition** est l’instruction **if…else**…que l’on écrit de la façon suivante :

* On commence par le mot clé if suivi de la condition entre ( )
* Suivi des { } qui contiennent les instructions à exécuter si la condition est vraie

Chaque instruction devra être suivie d’un ;

* En option on peut compléter par le mot clé else suivi des { } qui contiennent les instructions à exécuter si la condition est fausse.

La **condition** devra être une **opération logique.**

* Qui renvoie une valeur de type boolean ( soit true soit false, soit 0 ou 1 )
* On utilise pour cela des opérateurs logiques de comparaison,

Remarque : 0 est considéré comme false et toute valeur différente de 0 comme true

On écrira la condition sous la forme **variable == valeur**

Par exemple ; if(variable ==2) {instructions ;}

*si la condition « variable vaut 2 » est vraie alors exécuter les*

*instructions.*

Ne pas confondre l’opération logique == avec le signe d’affectation =

**Un jeu de lumières à 8 LEDs en utilisant un tableau de variables et une boucle for**

//… entête déclarative

//….déclarer ici les variables et constantes globales

const int LED[8]= {2,3,4,5,6,7,8,9} ;

int vitesse=250 ;

void **setup**() {

for (int i=0 ; int i<8 ; i++) {

pinMode(LED[i], OUTPUT);

} //… fin de for

} //…fin de setup()

void **loop**() {

for (int numLED=0 ; int numLED<8 ; numLED++) { //…défile les 8 LEDs

// …allume la LED voulue

for (int i=0; int i<8; i++) { //…répète 8 fois le code

if ( i==numLED) { //… si la LED est la LED courante

digitalWrite(LED[i], HIGH);

}

else { // … sinon

digitalWrite(LED[i], LOW);

}

} //…..fin de for i

delay(vitesse);

} //… fin de for numLED

} //…fin de la fonction loop()

**Présentation du Digit :**

Le digit à LEDs est un « afficheur » à chiffre rouge ou vert que l’on trouve sur les réveils ou autre appareils de mesure.

Un digit à LEDs n’est rien d’autre en fait que 8 LEDs de forme allongée mises dans un boitier et disposées en « 7 segments » + 1 point pour pouvoir afficher des chiffres.

Un digit aura logiquement :

* Une broche par LED ( 7 segments et 1 point ),

soit 8 broches,

* Une broche commune, qui va en fait être dédoublée

soit 2 broches,

* Un digit aura donc 10 broches en tout.

On distingue 2 types de digit suivant la broche qui est

commune :

* A cathode commune si - commun,
* A anode commune si + commun.

**Afficher les segments d’un digit : le montage**

* On va connecter chaque LED-segment du digit

à une broche Arduino via une résistance en série

pour chaque,

* On va connecter la cathode commune au 0v

On utilise les broches 2,3,4,5,6,7,8,9

Broche 8 🡺 LED a

Broche 9 🡺 LED b

Broche 4 🡺 LED c

Broche 3 🡺 LED d

Broche 2 🡺 LED e

Broche 7 🡺 LED f

Broche 6 🡺 LED g

Broche 5 🡺 point

**Afficher les segments d’un digit: le programme 1ère version**

On va allumer puis éteindre successivement tous les segments et le point du digit. On reprend une structure de programme que l’on a déjà utilisée pour réaliser un jeu de lumière avec les 8 LEDs.

Entête déclarative :

On déclare :

* Un tableau de 8 constantes appelé LED de type int pour désigner les 7 broches de « segments » ainsi que la broche « point ».
* Une variable de type int pour fixer la vitesse, appelée vitesse.

Fonction setup() :

A ce niveau on va ;

* Initialiser les broches utilisées en sortie en utilisant une boucle for, sous la forme LED[i] où i est la variable de comptage de la boucle.

Fonction loop() :

A ce niveau on va :

* A l’aide d’une boucle allumer successivement la LED voulue puis l’éteindre après une pause avant de passer à la boucle suivante,
* Le code de la fonction loop() se répète sans fin.

// …. Entête déclarative, on déclare ici les constantes et variables globales

Const int LED[8] = {8,9,4,3,2,7,6,5} ; // …Tableau des constantes

// … ordre des broches a,b,c,d,e,f,g,pt

Int vitesse=200 ; // …variable fixant la durée de la pause

// … la fonction setup() exécutée au début et en une seule fois

void **setup**() {

for (int i=0 ; i<8 ; i++) ; // ….répète 8 fois le code

pinMode(LED[i], OUTPUT) ; // …met la broche en sortie

} // ….fin de for

} //… fin de la fonction setup()

//…. La fonction loop() exécutée ensuite en boucle sans fin

void **loop**() {

// ….met les 8 broches en sortie

for ( int i=0 ; i<8 ; i++) { // ….répète 8 fois le code

digitalWrite(LED[i], HIGH) ; // ….allume la LED

delay(vitesse) ; // ….pause

digitalWrite(LED[i], LOW) ; //….éteint la LED

} //….fin de for

} // ….fin de la fonction loop()

**Afficher les segments d’un digit : le programme version améliorée :**

On peut rendre un tel programme plus visuel en utilisant des expressions binaires.

Entête déclarative

On déclare :

* Un tableau de 8 constantes appelé LED de type int pour désigner les 7 broches de segments ainsi que la broche du point.
* Une variable de type int pour désigner la vitesse appelée « vitesse ».

Fonction setup()

A ce niveau on va :

* Initialiser les broches utilisées en sortie en utilisant une boucle sous la forme LED[i] où i est la variable de comptage de la boucle.

Fonction loop()

A ce niveau on va :

* à l’aide d’une boucle allumer successivement la LED voulue ,
* Le code de la fonction loop se répète sans fin.

Puis une autre fonction : fonction de gestion des LEDs

On va ici séparer le code de gestion des LEDs dans une fonction séparée :

* Cette fonction va recevoir la valeur binaire,
* On défile les 8 broches et à chaque fois on met la LED dans l’état du 0/1 correspondant. On utilise pour cela une fonction du langage Arduino qui permet de lire la valeur d’un bit d’une variable bitRead(valeur,bit).
* Puis faire une pause entre 2 appels de la fonction.

//…entête déclarative

//….déclarer ici les constantes et variables globales

Const int LED[8] = {8,9,4,3,2,7,6,5} ; //…Tableau des constantes

// …ordre des broches : a,b,c,d,e,f,g,pt

int vitesse=1000 ; //…variable fixant la durée de la pose

//…..la fonction setup() : exécutée une fois et 1 seule fois

void **setup**() {

//…met les 8 broches en sortie…

for (int i=0 ; i<8 ; i++) { //… répète 8 fois le code

pinMode(LED[i], OUTPUT) //…met la broche en sortie

} //…fin de for

} //…fin de la fonction setup()

//…la fonction loop() : exécutée ensuite en boucle sans fin

void **loop**() {

//… ordre des broches : a,b,c,d,e,f,g,pt

allumeLED(B10000000) ; // …segment a

allumeLED(B01000000) ; //….segment b

allumeLED(B00100000) ; //….segment c

allumeLED(B00010000) ; //….segment d

allumeLED(B00001000) ; //….segment e

allumeLED(B00000100) ; //….segment f

allumeLED(B00000010) ; //…segment g

allumeLED(B00000001) ; //…segment pt

allumeLED(B00000000) ; // …;éteint tout

} //….fin de la fonction loop()

//…fonction qui reçoit une valeur de type int non signée et ne renvoie rien

void allumeLED(unsigned int valeur) { //….fonction pour allume/éteindre les LEDs

for (int i=0 ; i<=7 ; i++) {

digitalWrite(LED[i], bitRead(valeur,7-i)) ;

} //…fin de for

delay(vitesse); //…. Pause

} //…fin de la function allumeLED

**Principe d’affichage des chiffres sur un digit**

Voici le principe à utiliser :

* Pour afficher le 1 on allumera les segments b et c, les autres restant éteints,
* Pour afficher le 2 on allumera les segments a,b,d,e,g, les autres sont éteints
* Et ainsi de suite.

**Afficher les chiffres sur un digit : 1ère version**

Entête déclarative

On déclare :

* Un tableau de 8 constantes appelées LED de type int pour désigner les 7 broches de segments ainsi que la broche du point,
* Une variable de type int pour fixer la vitesse appelée « vitesse ».

Fonction setup()

A ce niveau on va :

* Iniyialiser les broches utilisées en sortie en utilisant une boucle, sous la forme LED[i] où est la variable de comptage de la boucle.

Fonction loop()

* A l’aide d’une boucle allumer successivement les LEDs voulues :

¤ B01100000 pour le 1

¤ B01101101 pour le 2

¤ B11110010 pour le 3 etc….

* Le code de la fonction loop() se répète sans fin….

Autre fonction : fonction de gestion des LEDs

On va ici séparer le code de fonction des LEDs dans une fonction séparée :

* Cette fonction va recevoir la valeur binaire,
* On défile les 8 broches et à chaque fois on met la LED dans l’état du 0/1 correspondant. On utilise pour cela une fonction du langage Arduino qui permet de lire la valeur d’un bit d’une variable, bitRead(valeur, bit),
* Puis faire une pause entre 2 valeurs de la fonction.

const int LED[8] = {8,9,4,3,27,6,5} ;

int vitesse = 1000 ;

void **setup**() {

for (int i=0 ; i<8 ; i++) {

pinMode(LED[i], OUTPUT) ;

} // fin de for

} // fin de setup()

void **loop**() {

allumeLED(B11111100) ; //….nombre 0

allumeLED(B01100000) ; //…nombre 1

allumeLED(B11011010) ; //…nombre 2

allumeLED(B11110010) ; //…nombre 3

allumeLED(B01100110) ; //…nombre 4

allumeLED(B10110110) ; // …nombre 5

allumeLED(B10111110) ; //…nombre 6

allumeLED(B11100000) ; //…nombre 7

allumeLED(B11111110) ; //…nombre 8

allumeLED(B11110110) ; //…nombre 9

allumeLED(B00000000) ; //…éteint tout

} //…fin de la fonction loop()

void allumeLED(unsigned int valeur) {

for (int i=0; i<=7; i++) {

digitalWrite(LED[i], bitRead(valeur, 7-1));

}

delay(vitesse);

} //…fin de la function allumeLED

**Utilisation d’un buzzer piézo-électrique et son utilisation avec Arduino**

Pour transformer une onde électrique en onde sonore on utilise un haut parleur.

Présentation du buzzer piezo-éléctrique :

Ce type de buzzer dispose d’une membrane métallique qui va vibrer sous l’action d’un signal électrique et va fournir un son de fréquence correspondante.

Le principe d’utilisation est très simple :

* On connecte le fil noir au 0V,
* On connecte le fil rouge sur la borne utilisée pour générer la fréquence du son.

Les caractéristiques de la capsule piezo-électrique sont les suivantes :

* Intensité = 2 mA
* Tension = 3 à 30V

**Produire un son simple avec une carte Arduino**

Pour comprendre

* Il faut produire une onde électrique de la fréquence voulue au buzzer,
* Si on met une broche E/S de la carte Arduino au niveau HAUT puis au niveau BAS et que l’on recommence rapidement avec un délais court entre chaque niveau, on va produire une onde carrée,
* C’est un peu comme si on faisait clignoter très vite le buzzer. Le problème va être de fixer la fréquence du son ce qui va se faire très simplement en fixant une pause en microsecondes à l’aide de (instruction delayMicroseconds() ).

Fonction setup()

* On commence par mettre en sortie la broche que l’on utilise.

Fonction loop()

De la même façon que pour faire clignoter une LED on va :

* Mettre la broche au niveau HAUT
* Faire une courte pause à l’aide de l’instruction delayMicroseconds() (pour 50Hz, utiliser 1000 µs = 1 ms),
* Mettre la broche au niveau BAS,
* Faire à nouveau une courte pause de même durée pour obtenir une onde carrée symétrique.

Remarque : La fréquence obtenue vaudra f=1/T = 10 / (2xdelai)…delai en µs

A l’inverse pour obtenir une fréquence donnée on utilisera une pause de :

Délai = 10 / (2xf) µs soit pou 500Hz, délai= 10 / (2x500) = 1000 µs

const int PIEZO = 2 ;

int demiPeriode = 1000 ;

void **setup**() {

pinMode(PIEZO, OUTPUT) ;

}

void **loop**() {

digitalWrite(PIEZO, HIGH) ;

delayMicroseconds(demiPeriode) ;

digitalWrite(PIEZO, LOW) ;

delayMicroseconds(demiPeriode) ;

}

**Produire un son de fréquence voulue avec une carte Arduino**

Devoir calculer la demi période c’est compliqué….on va simplifier…

Les instructions tone() et noTone()

Le langage Arduino propose une instruction dédiée qui permet de générer simplement une impulsion carrée de fréquence voulue sur n’importe quelle broche :

l’instruction tone(broche, fréquence)

Le son durera tant que l’instruction noTone(broche) n’aura pas été appelée.

Fonction setup()

On peut mettre la broche en sortie mais on n’est pas obligé de le faire parce que la fonction tone le fait automatiquement.

Fonction loop()

On appele tout simplement la fonction tone :

* En précisant la broche à utiliser,
* La fréquence voulue
* ….et c’est tout.
* Ensuite mettre une pause de durée voulue,
* Et stopper le son avec noTone(),
* Faire une nouvelle pause…..etc….

const int PIEZO=2 ;

int LA3=440 ; //…variable de la fréquence

void **setup**() {

pinMode(PIEZO, OUTPUT) ;

}

void **loop**() {

tone(PIEZO, LA3) ;

delay(1000) ;

noTone(PIEZO, LA3) ;

delay(1000);

}

**Produire une simple mélodie avec une carte Arduino**

Il devient facile de produire de petites alarme sonores ou même des mélodies simples

Variable de notes

L’idée est de créer des variables de notes dont le nom correspond à la note. Il ne reste plus qu’à créer un listing des notes et créer toutes les variables de notes….

Par exemple pour créer les 3 notes DO, RE, MI :

* int DO = 262 ;
* int RE = 294 ;
* int MI= 330 ;

Entête déclarative

On va déclarer à ce niveau :

* une constante désignant la broche utilisée,
* des constantes de notes
* 1 tableau de notes et 1 tableau de durée de notes.

Fonction setup()

La fonction tone met automatiquement la broche en sortie.

Fonction loop()

* On appelle tout simplement la fonction tone pour chaque note au sein d’une boucle for en précisant la broche à utiliser et la fréquence voulue,
* Ensuite mettre une pause de durée voulue
* Et stopper les sons avec noTone(),
* Faire une nouvelle pause ….etc…

const int PIEZO=2 ;

int DO= 262 ;

int RE=294 ;

int MI=330 ;

const int nombreNotes=11 ;

int note[nombreNotes] = {DO,DO,DO,RE,MI,RE,DO,MI,RE,RE,DO} ;

int duree[nombreNotes]= {1,1,1,1,1,2,1,1,1,1,1} ;

void **setup**() {

pinMode(PIEZO, OUTPUT) ;

}

void **loop**() {

for ( int i=0 ; i<nombreNotes ; i++) {

tone(PIEZO, note[i]);

delay(500\*duree[i]);

noTone(PIEZO);

delay(100);

}

noTone(PIEZO);

delay(1000);

}

**Le concept de modulation de largeur d’impulsion (MLI):**

Rappel :

* Numérique : tout ou rien, pas de niveau intermédiaire,
* Analogique : variation progressive avec tous les niveaux intermédiaires.

Par exemple, dans le cas d’une lampe en numérique, la lampe est allumée ou éteinte et en analogique, la lampe peut voir sa luminosité varier d’aucune lumière à sa luminosité maximale.

Simuler de l’analogique avec une broche numérique :

En définissant une période de temps assez courte on allume la LED que pendant une partie de cette durée par ex 25% et on la laisse éteinte les 75% restant et l’on recommence ensuite très rapidement 🡺 on verra la LED allumée au quart de sa luminosité maximale.

( la tension moyenne au fil du temps vaudra 25% de 5V soit 1,25V )

Le concept de Modulation de Largeur d’Impulsion ou PWM :

En pratique :

* On appelle « duty-cycle » la période de temps de base qui correspond à 100%,
* On fixera la *largeur de l’impulsion*, « pulse width » de 0 à 100%,
* Ce qui en binaire s’exprimera par un octet dont la valeur sera comprise entre 0 (0%) et 255 (100%).

*Avec le langage Arduino on génèrera une telle impulsion grâce à l’instruction* analogWrite() *disponible seulement sur les broches dites PWM ( -3, -5, -6, -9, -10, -11 ).*

**Faire varier la luminosité d’une LED : le programme**

Les possibilité de la modulation de largeur d’impulsion sont nombreuses notamment :

* Faire varier la luminosité d’une LED,
* Faire varier la vitesse d’un moteur,
* Générer des variations de couleurs avec une LED.

On va faire varier la luminosité d’une LED de 0% à 100% et de 100% à 0%.

Entête déclarative :

* On va déclarer à ce niveau la constante désignant la broche utilisée,
* La variable désignant la vitesse de l’impulsion,
* La variable désignant la largeur de l’impulsion.

Fonction setup() :

* On configure la broche de sortie avec l’instruction pinMode

Fonction loop() :

* On initialise une première boucle « croissante » avec l’instruction for(int i=0 ; i<=100 ; i++),
* Au sein de cette première boucle on fera varier la largeur d’impulsion de 0% à 100% grâce à l’instruction largeur=map(i, 0,100,0,255),
* La modulation de la largeur d’impulsion est déclenchée par l’instruction analogWrite(broche, largeur),
* On fixera une pause grâce à l’instruction delay(vitesse).
* Ensuite on initialise une deuxième boucle « décroissante » for(int i=100 ; i>=0 ; i--),
* On fait varier la largeur d’impulsion largeur=map(i,0,100,0,255),
* On déclenche la modulation de la largeur de l’impulsion analogWrite(broche,largeur),
* On fixe à nouveau une pause… delay(vitesse).

Remarque : l’instruction map(valeur, minSource,maxSource,minDestination,maxDestination) permet de ré-échelonner une valeur simplement.

Utile ici pour convertir une largeur d’impulsion comprise entre 0% et 100% dans la valeur correspondante comprise entre 0 et 255.

// entête déclarative

// déclarer ici les constantes et variables globales

const int LED=3 ; // constante désignant la broche de la LED

int vitesse=20 ; // temps de pause en ms

int largeur=0 ; // variable pour la largeur d’impulsion

void **setup**() {

pinMode(LED, OUTPUT) ; // met la broche en sortie

} // fin de setup()

void **loop**() {

for ( int i=0 ; i<=100 ; i++) { // mise en boucle croissante

largeur=map(i,0,100,0,255) ; // ré-échelonne I (0-100%) vers (0-255)

analogWrite(LED, largeur); // génère l’impulsion PWM voulue sur la broche

delay(vitesse); // pause

} // fin de for i

for ( int i=100 ; i>=0 ; i--) { // boucle décroissante

largeur=map(i,0,100,0,255) ;

analogWrite(LED, largeur) ;

delay(vitesse);

} // fin de for i

} // fin de la fonction loop

**Fiche composant : Découvrir la LED multicolore RGB**

La LED :

La LED est un composant électronique qui sert à produire de la lumière.

* La LED standard 5mm est de couleur rouge, verte ou jaune,
* La **tension à ses bornes est constante à 1,5V** environ lorsqu’elle est en circuit,
* Son intensité de fonctionnement est de l’ordre de 20mA,
* La LED est une diode : elle a donc un sens de connexion.

Le montage type d’une LED standard en 5V consiste à :

* Connecter la LED en série avec une résistance ( La patte – de la LED devra être tournée vers le - ),
* La résistance a pour but de limiter l’intensité. Utiliser une valeur entre 200 et 300 Ohms ( en pratique, j’utilise 270 Ω / ¼ de watt ).

La LED multicolore dite « ‘RGB » ou « RVB »

* Une LED RVB pour Rouge-Vert-Bleu ( ou RGB pour Red-Green-Blue), n’est autre qu’un ensemble de 3 LEDs couleurs assemblées dans un même boitier,
* Une LED RGB aura donc 4 broches. La broche commune pourra selon les modèles être le + (anode commune) ou le – (cathode commune),
* Pour obtenir les couleurs unitaires, il suffira d’allumer la LED de la couleur voulue.

Utiliser la LED multicolore RVB avec une carte Arduino : le montage

Le principe d’utilisation de la LED multicolore est le même que celui d’une LED ou plutôt de 3 LEDs avec une broche commune :

* La broche commune sera connectée au 0V (LED RGB à cathode commune) ou au + 5V (LED RGB à anode commune),
* Chaque LED sera connectée sur une broche E/S de la carte Arduino configurée en sortie via une résistance pour fixer l’intensité circulant dans la LED.

**ATTENTION** : On va utiliser 3 broches E/S particulières ayant le signe ~ devant le numéro sur la carte Arduino : ces broches ont une fonction spéciale que nous allons utiliser, la modulation de largeur d’impulsion (MLI ou PWM en anglais).

Connecter la broche R sur la broche 3, la broche V sur la broche 5 et la broche B sur la broche 6

**Utiliser la LED multicolore RVB avec une carte Arduino : allumer les couleurs unitaires**

Entête déclarative :

On va déclarer à ce niveau :

* 3 constantes utilisant les broches utilisées 🡺 constint
* 3 constantes binaires, une pour chaque couleur 🡺 constboolean
* 1 variable fixant la durée de la pause 🡺 int vitesse

Fonction setup() :

* On configure les 3 broches en sortie avec l’instruction pinMode()

Fonction loop() :

* On commence par allumer la première LED pendant 1 seconde puis on l’éteint,
* On fait de même pour les autres LEDs,
* Puis on allume les LEDs 2 à 2 ce qui permet de visualiser des couleurs par combinaison des LEDs de couleur,
* Enfin on termine en allumant les 3 LEDs ce qui donne du blanc

Fonction de gestion des LEDs RGB :

Pour simplifier le code on crée une fonction pour gérer l’affichage des LEDs :

* La fonction ne renvoie rien = type void
* Reçoit 3 valeurs binaires d’activation des couleurs RVB,
* A l’aide de 3 valeurs on teste les paramètres reçus en entrée :

¤ si true , on allume la LED,

¤ si false, on éteint la LED,

* La pause est intégrée dans cette fonction.

Remarque : Si on utilise une LED à anode commune, il suffit de mettre !HIGH à la place de HIGH et !LOW à la place de LOW dans le code de la fonction.

// entête déclarative

// déclarer ici constantes et variables globales

constint ledR=3 ; // constante désignant la broche de la LED

constint ledV=5 ; // constante désignant la broche de la LED

constint ledB=6 ; // constante désignant la broche de la LED

constboolean R=true ; // constante binaire couleur R

constboolean V=true ; // constante binaire couleur V

constboolean B=true; // constante binaire couleur B

int vitesse=1000 ; // variable fixant la durée de la pause

void **setup**() {

pinMode(ledR, OUTPUT) ;

pinMode(ledV, OUTPUT) ;

pinMode(ledB, OUTPUT);

} // fin de la fonction setup()

void **loop**() {

// …couleurs de base….

ledRVB(R,0,0), ledRVB(0,0,0) ; // rouge puis éteint

ledRVB(0,V,0), ledRVB(0,0,0) ; // vert puis éteint

ledRVB(0,0,B), ledRVB(0,0,0) ; // bleu puis éteint

// ….mix….

ledRVB(R,V,0), ledRVB(0,0,0) ; // rouge + vert = jaune puis éteint

ledRVB(R,0,B), ledRVB(0,0,0) ; // rouge + bleu = violet puis éteint

ledRVB(0,V,B), ledRVB(0,0,0); // vert + bleu = bleu clair puis éteint

ledRVB(R,V,B), ledRVB(0,0,0); // rouge + vert + bleu = blanc puis éteint

} // fin de la fonction loop()

//……fonction pour combiner les couleurs ON / OFF …

void ledRVB(boolean Rouge, boolean Vert, boolean Bleu) {

if (Rouge==true) digitalWrite(ledR, HIGH);

else digitalWrite(ledR, LOW);

if (Vert==true) digitalWrite(ledV, HIGH);

else digitalWrite(ledV, LOW);

if (Bleu==true) digitalWrite(ledB, HIGH);

else digitalWrite(ledB, LOW);

delay (vitesse);

}