# Cahier de 25 exercices
# ARDUINO Niveau débutant

Nom : _____

Classe : _____

Platine n°: _____


Version du 19 août 2019 – Jean-Pierre Conil

# Présentation générale

## But de ce cahier

Ce cahier de 25 exercices est conçu pour s'initier progressivement à la programmation embarquée sur Arduino. Il est recommandé d'utiliser un Easy Shield V1. Chaque exercice est accompagné de 3 variations. La progression pédagogique est présentée au début et intéressera plus particulièrement les enseignants. Le site de l'auteur (www.passtek.net) permet d'accéder à des ressources gratuites (Posters, QCM, lien vers des vidéos). Chaque exercice est présenté sur une double page avec 6 rubriques : Programmer (le cahier des charges et le programme complet), Comprendre (les concepts abordés), Faire marcher (une check-list), S'entraîner (3 nouveaux exercices), Se souvenir, Noter et Mesurer (les nombres qui permettent de comparer les performances du programme). Les exercices proposent de façon graduelle des applications concrètes (carillon, récepteur de télécommande IR, sirène américaine, variateur de lumière et de couleur...) le plus souvent en moins de 10 instructions. On essaie de donner de bonnes habitudes de programmation.

## Comment l'utiliser ?

Une façon simple de l'utiliser est de recopier sans erreur le programme fourni et de le faire marcher puis de se lancer dans les variantes proposées. Si on a un groupe hétérogène on peut ne faire faire qu'une variante ou toutes pour compenser les différences de niveaux. Sur les premiers niveaux des vidéos sont disponibles sur Youtube.

# Définition des niveaux

## Débutant

On fait la découverte des composants simples du shield : LED, bouton-poussoir, LDR, buzzer, capteur de température, LED RGB et récepteur IR.

On utilise les fonctions delay(), tone(), notone(), random(). Les opérateurs % et ! sont compris.

Pour les Entrées/Sorties on utilise digitalWrite() puis digitalRead() et enfin analogRead() et analogWrite(). Les fonctions map() et analogReference() sont aussi vues.

Pour les variables on utilise seulement const, int et byte. On voit aussi les tableaux.

Les structures if et if else sont utilisées avec des conditions simples. La boucle for est utilisée et la notion de portée des variables est alors découverte.

On est capable de remplir un tableau de valeur et de faire une moyenne.

La communication série est utilisée avec le logiciel Regressi pour faire un suivi de grandeur physique.

On sait mesurer le temps d'une exécution de la fonction loop().

## Moyen

On utilise millis() et delay() de façon à avoir une période de boucle ($T_{LOOP}$) de l'ordre de 100ms. Les temporisations utilisent $T_{LOOP}$ comme base de temps.

Les types de variables unsigned, long et float sont maitrisées. On sait utiliser sizeof() notamment dans une boucle for.

On fait calculer le compilateur.

Les structures if, else, elseif, ainsi que while et do while sont utilisées avec des conditions doubles.

La liaison série est utilisée en réception de caractère unique pour piloter une application.

La notion d'hystérésis est mise en pratique.

On crée des fonctions simples void fonction(var). On introduit les fonctions logiques.

On commence à faire des commentaires simples.

## Confirmé

On utilise enfin les bibliothèques et avec elles le DHT11.

Tous les types de variable sont connus et la notion de portée est maîtrisée.

La structure switch est vue  avec une application avec la liaison série et une application dans un grafcet simple.

On sait utiliser les ports du µcontrôleur avec des fonctions bit-à-bit.

La création des fonctions avec renvoi de valeur est maîtrisée.

On utilise un afficheur LCD/ I²C et un servomoteur.

On sait structurer et présenter la structure de son programme. On commente aussi son programme de façon pertinente.

On utilise et gère les différents types de mémoire (disque, PROGMEM).

## Expert

On crée des bibliothèques.

On se sert des interruptions.

On hiérarchise les grafcets.

On utilise les structures et les pointeurs.

On optimise : la consommation ou le temps d'exécution ou la place en mémoire…

On sait faire ergonomique.

On connaît la récursivité (et on l'évite).

# Les outils

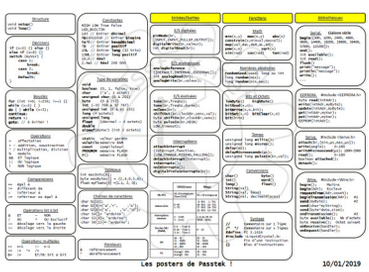
## L'IDE

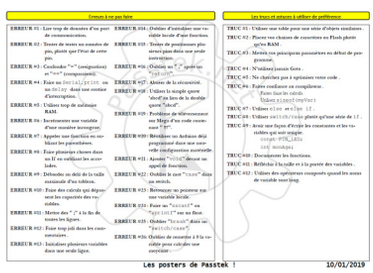
On utilise comme environnement de développement intégré (IDE) le logiciel Arduino. Il se trouve en téléchargement gratuit sur le site https://www.arduino.cc . Téléchargez et installez la dernière version ou mettez-la sur une clé USB. Le logiciel peut fonctionner directement depuis la clé.

Cet IDE fournit la coloration syntaxique ainsi que des fonctions de gestion des Entrées/Sorties biens commodes au début.

## Les posters

Je vous propose 3 posters au format A3. Un poster reprend toutes les instructions, les fonctions les plus courantes et la syntaxe. Il est inspiré d'une « cheat sheet ». Un deuxième poster fait la liste des erreurs à ne pas faire et des bonnes pratiques de programmation (donc à faire). Le troisième poster est une carte mentale décrivant la formation. Vous trouverez ces posters sur le site www.passtek.net , dans le cours Arduino et la rubrique « Documents et Liens ».






## Les vidéos

Chaque exercice du niveau débutant est accompagné d'une vidéo disponible sur la chaîne Passtek de YouTube. Pour les niveaux suivants les vidéos présentent des concepts et ne sont plus directement liées à un exercice en particulier. Il vous faudra choisir celles qui correspondent aux lacunes de l'apprenant ou au point que vous voulez mettre en valeur.

## Les QCM

La progression pédagogique est rythmée par des QCM pour Pronote que l'on peut utiliser de façon formative ou évaluative. Chaque QCM reprend les questions des QCM précédents. Pour voir quels concepts sont abordés dans les exercices et les QCM, il faut se reporter au tableau Progression pédagogique.

# Les mesures

## Intérêt

Je me suis demandé de combien de façons différentes je pouvais programmer un chenillard (une LED est éclairée dans une rangée de 4 LEDs et la LED éclairée parcours la rangée). Au bout de quelques jours j'avais plus de 15 façons différentes de programmer ce chenillard… Précisons que tous ces programmes avaient été écrits avec le même IDE et la même cible, un Arduino UNO.

Il est bien évident que toutes ces solutions ne sont pas équivalentes. Pour pouvoir les comparer il faut pouvoir mesurer ! Pour chaque programme on vous propose donc 4 grandeurs mesurables facilement de façon précise (et non subjective).

## Instructions

Pour déterminer le nombre d'instructions, il suffit de compter le nombre de point-virgule dans le programme. Nous avons choisi d'inclure les déclarations de constantes et de variables par simplicité.

Attention ce n'est pas le nombre de lignes qui lui aussi est très facile à déterminer.

## Flash

Lors de la compilation, l'IDE indique le nombre d'octets qu'occupe le programme  dans la mémoire Flash du µcontrôleur.

Cette mesure est légèrement biaisée par la place qu'utilise le bootloader (512 octets pour l'Arduino UNO).

## RAM

Lors de la compilation, l'IDE indique le nombre d'octets de RAM qui est utilisé par le programme pour les variables.

Vous constaterez rapidement que l'utilisation de la liaison série réclame 80 octets de RAM ou que la fonction delay() nécessite un certain nombre d'octets. Ce n'est pas un problème tant que le programme est simple.

## $T_{LOOP}$

C'est le temps qu'il faut pour que les instructions de la fonction principale loop() soient toutes exécutées et que l'on recommence. L'exercice 22 du niveau débutant explique comment on fait la mesure. Il faut un oscilloscope ou un fréquencemètre pour cela.

Cette mesure donne une idée de la rapidité du programme à effectuer les tâches prévues et à réagir à des évènements externes quand on n'utilise pas d'interruption.

# Sommaire
# La progression pédagogique

| N° | Concept | Thème | Évaluation |
|---|---|---|---|
| **1** | constantes<br>setup()<br>pinMode() digitalWrite()<br>; {} | Allumer la LED sur D13 | |
| **2** | loop()<br>const int<br>delay()<br>CONSTANTE | Faire clignoter la LED sur D13 | |
| **3** | digitalRead()<br>! | Allumer une LED quand on appuie sur un BP | |
| **4** | tone(pin,fréquence,durée)<br>if (condition) {action ;}<br>== | Jouer un LA(440Hz) quand on appuie sur un BP | QCM n°1 |
| **5** | byte | Bascule ou mémoire | |
| **6** | front montant<br>! | Télérupteur | |
| **7** | tableau<br>for<br>% | Chenillard | |
| **8** | i++<br>for (i=1000 ;i>500 ;i=i-1) {} | Sirène américaine | |
| **9** | tone(pin,fréquence) | Jouer un morceau de musique | |
| **10** | portée des variables | Carillon | QCM n°2 |
| **11** | analogRead()<br>Serial.begin(vitesse)<br>Serial.println(valeur) | Mesurer la tension sur le curseur d'un potentiomètre et envoyer le résultat à l'ordinateur | |
| **12** | if ( ) {act1 ;} else {act2}<br>noTone(pin)<br>division euclidienne | Régler la fréquence d'un son avec le potentiomètre | |
| **13** | map()<br>portée | Régler la fréquence d'un son avec le potentiomètre | |
| **14** | Régressi<br>Serial.print(temps++) | Courbe d'éclairement | |
| **15** | analogReference(INTERNAL)<br>LM35 | Thermostat | |
| **16** | analogReference(DEFAULT) | Thermostat avec seuil réglable | |
| **17** | analogWrite()<br>MLI/PWM | Variateur de lumière | |
| **18** | for (r=0 ;r<256 ;r=r+51){}<br>LED RGB | Générer des couleurs | |
| **19** | % | Programmer une couleur | |
| **20** | random() | Couleurs aléatoires | |
| **21** | != | Variateur de lumière blanche | QCM n°3 |
| **22** | PORTD<br>0b10000000 | Mesure de $T_{LOOP}$ | |
| **23** | tableau<br>index%=8 | Mesure de luminosité | |
| **24** | tableau<br>Organisation du programme | Horloge | |
| **25** | | Thermomètre lumineux | micro-projet |

# La platine Easy Module Shield V1

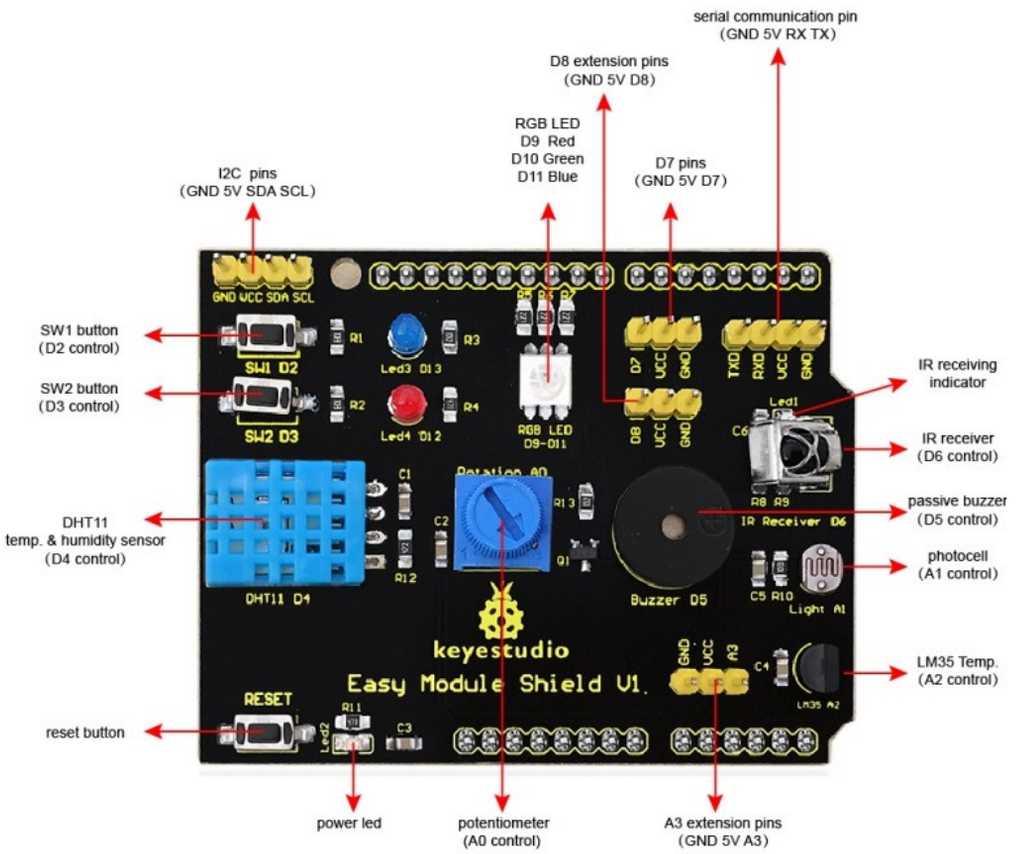
## Intérêt

La platine Easy Module Shield V1 permet d'utiliser rapidement des boutons-poussoir, des LEDs, des capteurs (température, lumière, IR) et de générer des sons sans avoir à réaliser de montage ou à se tracasser avec des connecteurs. Le coût est modique (environ 10€ pièce).

On installe la platine sur l'Arduino et on n'a plus qu'à programmer. Cela supprime la prise de tête du hardware !

Tous les programmes proposés dans ce livre d'exercices fonctionnent avec l'Easy Module Shield V1.

## Raccordement



# Tableau des E/S (entrées/Sorties)

**Entrées/Sorties digitales (ou Tout-Ou-Rien)**

| | |
|---|---|
| D2 | Bouton-Poussoir |
| D3 | Bouton-Poussoir |
| D4 | DHT11 Mesure température et hygrométrie |
| D5 | Buzzer |
| D6 | Récepteur IR (Infra-Rouge) |
| D7 | libre |
| D8 | libre |
| D9 | LED RGB (rouge) |
| D10 | LED RGB (vert) |
| D11 | LED RGB (bleu) |
| D12 | LED rouge |
| D13 | LED bleue (et LED sur la carte Arduino) |

**Entrées analogiques**

| | |
|---|---|
| A0 | Potentiomètre |
| A1 | LDR (niveau de luminosité) |
| A2 | LM35 (Température) |
| A3 | libre |

# Exercice 1

## Programmer :

**Allumer la LED branchée sur la broche 13.**

```
void setup() {
  pinMode(13,OUTPUT);
  digitalWrite(13,HIGH);
  }

void loop() {
  }
```

**Mesurer :**

2 instructions

Flash : 710 octets

RAM : 9 octets

$T_{LOOP}$ : 0 s

☐a ☐b ☐c

## Comprendre :

**OUTPUT, HIGH** // Ce sont des constantes. On ne peut pas les changer

// Le logiciel Arduino remplace automatiquement ces mots par la valeur 1.

**void setup() {}** // les instructions sont exécutées une seule fois à la mise sous tension ou après avoir appuyé sur le bouton Reset.

**pinMode(13,OUTPUT ) ;** // permet de choisir si la broche 13 (pin en anglais) est une entrée (INPUT) ou une sortie (OUTPUT). Ici c'est une sortie vers une LED.

**digitalWrite(13,HIGH) ;** // établit le niveau (HIGH ou LOW) d'une broche définie en sortie. Si on met HIGH la LED est allumée.

**; {}** // Toutes les instructions se terminent par un point-virgule ;

// On peut grouper plusieurs instructions entre deux accolades {}, mais elles sont quand même terminées par un point-virgule.

***Indentation*** // C'est le décalage du début de certaines lignes pour rendre le programme plus lisible

## Faire marcher :

- ☐ Ne pas oublier les **;** à la fin de chaque instructions !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Allumer d'autres LED (les broches 12, 11, 10 et 9 sur l'Easy Shield).

b) Allumer plusieurs LED à la fois.

c) Déterminer l'ensemble des couleurs que l'on peut générer de cette façon avec la LED RGB.

## Se souvenir :

void setup() { }

pinMode(broche, OUTPUT) ;

digitalWrite(broche,HIGH);

; {}

## Noter :

# Exercice 2

## Programmer :

**Faire clignoter la LED branchée sur la broche 13.**

```
const int PIN_LED = 13;
const int TEMPO = 500;

void setup() {
  pinMode(PIN_LED,OUTPUT);
}

void loop() {
  digitalWrite(PIN_LED,HIGH);
  delay(TEMPO);
  digitalWrite(PIN_LED,LOW);
  delay(TEMPO);
}
```

**Mesurer :**

7 instructions

Flash : 928 octets

RAM : 9 octets

$T_{LOOP}$ : 1 s

☐a ☐b ☐c

## Comprendre :

**void loop() {}** // Les instructions sont exécutées les unes après les autres, quand le µcontroleur arrive à la dernière il recommence au début. La valeur $T_{LOOP}$ est le temps qu'il faut pour faire toutes les instructions.

**const int PIN_LED = 13 ;** // const est l'abréviation de constante et int de integer (entier codé sur 2 octets). Au lieu d'utiliser 13 on va utiliser PIN_LED dans les fonctions pinMode et digitalWrite .

// On écrit les noms des constantes en MAJUSCULES et on sépare les mots par des tirets bas (_).

**delay( TEMPO) ;** // Le µcontrôleur attend « TEMPO » millisecondes avant de passer à l'instruction suivante.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instructions !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Allumer deux LED en alternance.

b) Créer un flash 0,1 seconde et de fréquence 1 Hz.

c) Faire un mini feu tricolore avec la LED RGB.

## Se souvenir :

void loop(){} ;

const int

delay() ;

## Noter :

# Exercice 3

## Programmer :

**Allumer une LED quand on appuie sur un bouton-poussoir.**

**Mesurer :**
5 instructions
Flash : 896 octets
RAM : 9 octets
$T_{LOOP}$ : 1 s
☐a ☐b ☐c

```
const int PIN_LED = 12;
const int PIN_BP = 6;

void setup() {
  pinMode(PIN_LED,OUTPUT);
  pinMode(PIN_BP,INPUT);
}

void loop() {
  digitalWrite(PIN_LED, !digitalRead(PIN_BP));
}
```

## Comprendre :

**digitalRead(PIN_BP );** // Lit le niveau sur la broche PIN_BP et renvoie 1 si il y 5 V et 0 si c'est 0 V.

// Quand le BP est au repos il y a 5 V sur la broche.

**!** // On **inverse** la valeur logique. Si c'est 1 cela devient 0 et si c'est 0 ça devient 1.

// C'est une **opération** logique de base : l'opérateur (ou fonction) **NON**.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instructions !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Allumer deux LED avec 2 BPs.

b) Essayer avec le capteur IR (broche 6).

c) Essayer sans l'inversion (le !).

## Se souvenir :

digitalRead(broche) ;

!

## Noter :

# Exercice 4

## Programmer :

**Jouer un LA (440 Hz) quand on appuie sur un bouton-poussoir.**

```
const int PIN_BUZZER = 5;
const int PIN_BP = 2;
const int LA = 440;
const int TEMPO = 100;

int bp = 0;

void setup() {
   pinMode(PIN_BP,INPUT);
}

void loop() {
  bp = !digitalRead(PIN_BP);
  if (bp == 1){
    tone(PIN_BUZZER,LA,TEMPO);}
}
```

**Mesurer :**

8 instructions

Flash : 1652 octets

RAM : 34 octets

$T_{LOOP}$ :

☐a ☐b ☐c

## Comprendre :

**tone(PIN_BUZZER,LA,TEMPO);**

// Génère un signal carré sur la broche PIN_BUZZER, de fréquence LA et de durée TEMPO.

// Si on écrit tone(PIN_BUZZER, LA) le son ne s'arrêtera que quand on utilisera la fonction noTone() .

**if (bp==0) {action1;}**

// Il faut **toujours** écrire la comparaison entre des parenthèses.

// L'opérateur de comparaison est **==** et non pas = (qui est une affectation).

## Faire marcher :

- ☐ Ne pas oublier les **;** à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Avoir 2 notes différentes ( LA et DO (262 Hz)) avec les 2 BPs (c'est un piano à 2 touches ☺).

b) L'appui sur le BP provoque aussi l'allumage d'une LED.

c) L'appui sur le BP provoque le jeu de 2 notes successives (on commence une mélodie...

## Se souvenir :

tone(broche,Hz) ;

if (condition) {action;}

Les opérateurs de comparaison sont : **== != < <= > >=**

## Noter :

# Exercice 5

## Programmer :

**Quand on appuie sur un bouton-poussoir la LED s'allume et reste allumée. Quand on appuie sur l'autre BP la LED s'éteint.**

**Mesurer :**
11 instructions
Flash : 964 octets
RAM : 10 octets
$T_{LOOP}$ : 12,5 µs
☐a ☐b ☐c

```
const int PIN_BUZZER = 5;
const int PIN_BP1 = 2;
const int PIN_BP2 = 3;
const int PIN_LED = 13;
byte bascule = 0;

void setup() {
   pinMode(PIN_BP1,INPUT);
   pinMode(PIN_BP2,INPUT);
   pinMode(PIN_LED,OUTPUT);
}

void loop() {
  if (!digitalRead(PIN_BP1)){ bascule=1;}
  if (!digitalRead(PIN_BP2)){ bascule=0;}
  digitalWrite(PIN_LED,bascule);
}
```

## Comprendre :

**byte bascule = 0;** // On déclare une variable (un emplacement en mémoire) de type byte (Fr: octet. La valeur de bascule est donc comprise entre 0 et 255.

// Les variables de type byte sont moitié plus petite que les variables de type int.

**if (!digitalRead(PIN_BP1)){ bascule=1;}**

// La condition est équivalente à (digitalRead(PIN_BP1)==0). En fait si la condition est fausse, elle a la valeur 0. Une variable (ou le résultat d'un calcul ou une fonction) peut être utilisée comme condition. La condition sera vraie dès que la variable sera différente de 0.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Commander 2 LED ( en simultané ou en inversé).

b) Émettre un Beep (durée 0,1s) lors de la mise en marche.

c) Émettre un Beep lors de l'appui sur les BP.

## Se souvenir :

byte

if (condition) {action;}

condition : cela peut être une variable, un calcul, le résultat d'une fonction.

## Noter :

# Exercice 6

## Programmer :

**Un télérupteur. Chaque fois que l'on appuie sur un bouton-poussoir la LED change d'état.**

**Mesurer :**
12 instructions
Flash : 1076 octets
RAM : 12 octets
$T_{LOOP}$ : 0,2 s
☐a ☐b ☐c

```
const int PIN_BP = 2;
const int PIN_LED = 13;
byte bpAvant = 0;
byte led =0;
byte bp = 0;

void setup() {
   pinMode(PIN_BP,INPUT);
   pinMode(PIN_LED,OUTPUT);
}

void loop() {
  bpAvant = bp;
  bp = !digitalRead(PIN_BP);
  if (bp==1){                  // le BP est appuyé
    if (bpAvant==0){           // et juste avant il ne l'était pas
      led = !led;
      digitalWrite(PIN_LED,led);
      delay(200);              // Rebond
      }
    }
}
```

## Comprendre :

**digitalWrite(PIN_LED,led);** // Eteint la LED si la variable led vaut 0, l'éclaire sinon. On a utilisé une variable plutôt que HIGH et LOW.

**bp = !digitalRead(PIN_BP);** // la fonction logique NOT ( ! ) donne 0 si la valeur à inverser est différente de 0 et donne 1 si la valeur à inverser est 0.

**delay(200) ;** // Cette temporisation évite la prise en compte des rebonds du BP.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Commander 2 LED ( en simultané ou en inversé).

b) Avoir 2 BPs qui commandent 2 LED indépendamment.

c) Émettre un Beep lors de l'éclairage.

## Se souvenir :

byte Octet = 8 bits (de 0 à 255)

! Opérateur logique NOT

FALSE vaut 0

TRUE pour toutes les valeurs différentes de 0

## Noter :

# Exercice 7

## Programmer :

**Un chenillard à 5 LED. Une seule est allumée à la fois pendant 0,5s.**

**Mesurer :**
8 instructions
Flash : 1088 octets
RAM : 16 octets
$T_{LOOP}$ : 0,5 s
☐a ☐b ☐c

```
const int TEMPO = 500;
byte led[] = {13,12,11,10,9};
byte i = 0;

void setup() {
  for (i=0;i<5;i=i+1){
   pinMode(led[i],OUTPUT);
  }
}

void loop() {
  digitalWrite(led[i],LOW);
  i = (i+1)%5;
  digitalWrite(led[i],HIGH);
  delay(TEMPO);
}
```

## Comprendre :

**byte pinLed[] = {13,12,11,10,9};** // On définit un tableau de variables de type byte. Ici ce sont les broches des 5 LED. La valeur de pinlLed[0] est 13. Voir le mémento pour les différentes façons de déclarer un tableau.

**for (i=0;i<5;i=i+1) { pinMode(led[i],OUTPUT); }**

// L'action est faite pour i =0, 1, 2, 3, 4

**i = (i+1)%5;** // Le calcul est le suivant : on ajoute 1 à i, puis on divise par 5 et on garde le reste de la division. Le résultat est forcément compris entre 0 et 4. **% se lit MODULO**

**// %** donne le reste de la division euclidienne.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Changer le sens du chenillard.

b) Changer le sens du chenillard quand on appuie sur un BP (maintenu ou bref).

c) Changer la temporisation du chenillard quand on appuie sur un ou deux BP.

## Se souvenir :

tableau[]

% Opérateur MODULO

for ( i=MIN; i<MAX; i=i+1) { action; }

## Noter :

# Exercice 8

## Programmer :

**Une sirène américaine. La fréquence émise par le buzzer passe de 500 à 1000 Hz en 0,5s puis de 1000 à 500 Hz en 0,5s.**

**Mesurer :**
6 instructions
Flash : 2384 octets
RAM : 34 octets
$T_{LOOP}$ : 1 s
☐a ☐b ☐c

```
const byte PIN_BUZZER = 5;
int i = 0;

void setup() {
  for (i=500; i<1000; i=i+1) {
    tone(PIN_BUZZER, i);
    delay(1);
  }
  for (i=1000; i>500; i=i-1) {
    tone(PIN_BUZZER, i);
    delay(1);
  }
  noTone(PIN_BUZZER);
}

void loop() {
}
```

## Comprendre :

**i++** // C'est la même chose qu'écrire i = i + 1. On incrémente i de 1.

**for (i=1000;i>500;i=i-1) { action; }**

// La boucle for se fait aussi de façon décroissante.

// On pourrait choisir un pas plus grand : i = i -2 ou i = i + 2

// Attention à mettre les choses dans le bon ordre sinon la boucle for risque de durer beaucoup plus longtemps. Ex : for (i=1000;i>500;i=i+1) { action; } va faire mal aux oreilles !

## Faire marcher :

- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Ne pas oublier les **;** à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Commander la sirène avec un BP. Changer les valeurs littérales des fréquences par des constantes. Mettre le programme dans la fonction loop().

b) Faire clignoter les deux LED bleues alternativement quand la sirène fonctionne.

c) Essayer d'autres pas pour la variation de la fréquence. Créer un variable pas de type int ou byte et remplacer le 1 dans le programme.

## Se souvenir :

i++ (et i--)

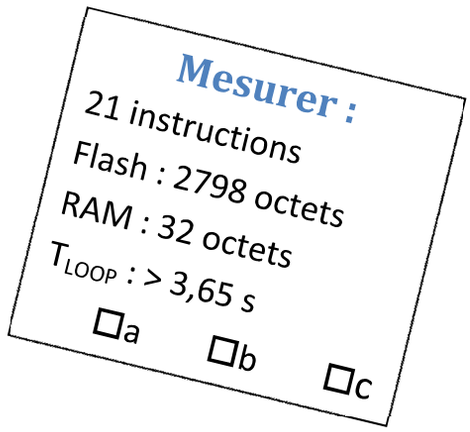
for ( i=MAX; i>MIN; i=i-1) { action; }

## Noter :

# Exercice 9

## Programmer :

**Jouer un morceau de musique.**




```
const byte PIN_BUZZER = 5;
const int f = 349;
const int a = 440;
const int cH = 523;

void setup() {
  tone(PIN_BUZZER,a,450);
  delay(500);
  tone(PIN_BUZZER,a,450);
  delay(500);
  tone(PIN_BUZZER,a,450);
  delay(500);
  tone(PIN_BUZZER,f);
  delay(350);
  tone(PIN_BUZZER,cH);
  delay(150);
  tone(PIN_BUZZER,a);
  delay(500);
  tone(PIN_BUZZER,f);
  delay(350);
  tone(PIN_BUZZER,cH);
  delay(150);
  tone(PIN_BUZZER,a,650);
 }

void loop() {
  }
```

## Comprendre :

**tone(PIN_BUZZER,a,450);** // La note a va être jouée pendant 450 ms puis le son s'arrête de lui-même sans qu'on ait à utiliser noTone()

const int a = 440; // On ne peut pas utiliser le type byte car la valeur est supérieure à 255.

void setup() // La mélodie est mise dans la fonction setup() pour n'être jouée qu'une seule fois au démarrage.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Commander la musique avec un BP.

b) Mettre les notes et les durées dans deux tableaux. Laisser 50ms de silence entre chaque note.

c) Faire flasher une LED à chaque nouvelle note.

## Se souvenir :

tone(PIN,note,**duree**);

La différence entre byte et int

## Noter :

# Exercice 10

## Programmer :

**Une sonnette de maison. Lorsqu'on appuie sur un BP une LED rouge s'allume et une mélodie est jouée 3 fois (avec une pause de 4s ).**

**Mesurer :**

28 instructions

Flash : 3058 octets

RAM : 32 octets

$T_{LOOP}$ : > 15 s

☐a ☐b ☐c

```
const byte PIN_BUZZER = 5;
const byte PIN_BP = 2;
const byte PIN_LED = 12;
const int f = 349;
const int a = 440;
const int cH = 523;

void setup() {
  pinMode(PIN_BP,INPUT);
  pinMode(PIN_LED,OUTPUT);
 }

void loop() {
  if (!digitalRead(PIN_BP)){
    digitalWrite(PIN_LED,HIGH);
    for (int i=0; i<3; i++){
      tone(PIN_BUZZER,a,450);
      delay(500);
      tone(PIN_BUZZER,a,450);
      delay(500);
      tone(PIN_BUZZER,a,450);
      delay(500);
      tone(PIN_BUZZER,f);
      delay(350);
      tone(PIN_BUZZER,cH);
      delay(150);
      tone(PIN_BUZZER,a);
      delay(500);
      tone(PIN_BUZZER,f);
      delay(350);
      tone(PIN_BUZZER,cH);
      delay(150);
      tone(PIN_BUZZER,a,650);
      delay(3350);
    }
    digitalWrite(PIN_LED,LOW);
  }
}
```

## Comprendre :

**for (int i=0; i<3; i++){action;}**

// L'action va être effectuée 3 fois.

// Comme la variable i est déclarée dans la () et elle ne peut être utilisée que dans la boucle for.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Comme pour l'exercice précédent mettre les notes et les durées dans deux tableaux. Laisser 50ms de silence entre chaque note.

b) Pour quelqu'un de malentendant faire flasher la LED bleue pendant la musique.

c) Faire jouer les 3 mélodies de plus en plus vite.

## Se souvenir :

for (int i=0; i<3; i++){action;}

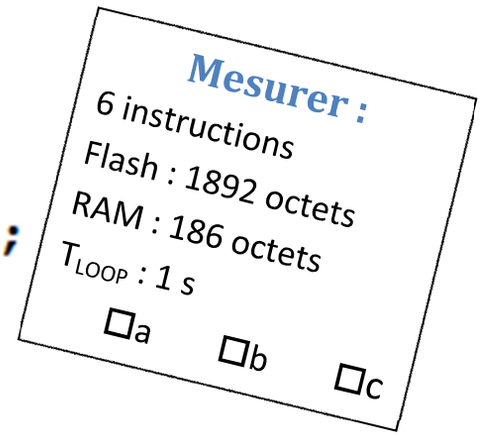
La différence entre byte (1 octet) et int (2 octets)

## Noter :

# Exercice 11

## Programmer :

**Mesurer la tension d'un potentiomètre et envoyer le résultat à l'ordinateur.**




```
const byte PIN_POT = 0;

void setup() {
  Serial.begin(115200);
 }

void loop() {
  int valeur = 0;
  valeur = analogRead(PIN_POT);
  Serial.println(valeur);
  delay(1000);
}
```

## Comprendre :

**Serial.begin(115200);** // On configure une liaison série vers l'ordinateur avec le débit de 115 200 bits/s. C'est le débit maximal. Dans le menu Outils il faut ouvrir le moniteur série. La partie **Serial.** indique que la fonction **begin** se trouve dans la bibliothèque Serial.

**valeur = analogRead(PIN_POT);** // La tension présente sur la broche PIN_POT est transformée en une valeur comprise entre 0 (pour 0 V) et 1023 (pour 5 V ou Vcc). C'est une Conversion Analogique Numérique sur 10 bits. Cette valeur est stockée dans la variable valeur.

**Serial.println(valeur);** // valeur est envoyé à l'ordinateur par la liaison série et s'affiche dans le moniteur série. Comme on a print**ln** les caractères CR (Retour Chariot) et LF (Nouvelle Ligne) sont rajoutés à la fin et provoque le retour à la ligne.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Mesurer la valeur quand c'est la LDR. Déterminer les valeurs minimale et maximale.

b) Mesurer la valeur quand c'est le capteur de température LM35. Chauffer le capteur et observer l'évolution du résultat.

c) Au lieu de println essayer avec print. Rajouter une virgule entre chaque envoi. Utiliser print(",").

## Se souvenir :

```
Serial.begin(115200);


Serial.println(valeur);

valeur = analogRead(broche);
```

## Noter :

# Exercice 12

## Programmer :

**Régler la fréquence du buzzer avec le potentiomètre et émettre le son seulement si on appuie sur un BP.**

**Mesurer :**
9 instructions
Flash : 2390 octets
RAM : 32 octets
$T_{LOOP}$ : 315 µs
☐a ☐b ☐c

```
const byte PIN_BUZZER = 5;
const byte PIN_BP = 2;
const byte PIN_POT = 0;
const int FREQ_MIN = 440;

void setup() {
  pinMode(PIN_BP,INPUT);
 }

void loop() {
  int valeur = 0;
  valeur = FREQ_MIN + analogRead(PIN_POT)/2;
  if (!digitalRead(PIN_BP)){
    tone(PIN_BUZZER,valeur);
  }
  else {
    noTone(PIN_BUZZER);
  }
}
```

## Comprendre :

**valeur = FREQ_MIN + analogRead(PIN_POT)/2;**

// Ces calculs sont nécessaires pour adapter la valeur lue avec le buzzer. Ils sont faits sur des valeurs entières et donnent des résultats entiers (5 divisé par 2 donne 2). Il faudra utiliser un autre type de variable pour les nombres décimaux.

// Les divisions et les multiplications par des puissances de 2 (2, 4, 8, 16, 32, 64, 128…) demandent beaucoup moins « d'efforts » au microcontrôleur. Quand c'est possible c'est celles qu'on utilisera.

**(!digitalRead(PIN_BP))** // On inverse la valeur lue sur la broche car le BP met la broche à la masse. Comme il y a une opération on est obligé d'utiliser des parenthèses.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Faire varier la fréquence du buzzer entre 200 Hz et 2 kHz.

b) Utiliser un BP pour enregistrer la valeur du potentiomètre et un autre pour émettre le son.

c) Envoyer la fréquence jouée par la liaison série quand on appuie sur un BP.

## Se souvenir :

Calculs entiers

Puissances de 2

## Noter :

# Exercice 13

## Programmer :

**Régler la fréquence du buzzer avec le potentiomètre et émettre le son seulement si on appuie sur un BP.**

**Mesurer :**
9 instructions
Flash : 2484 octets
RAM : 36 octets
$T_{LOOP}$ : 365 µs
☐a ☐b ☐c

```
const byte PIN_BUZZER = 5;
const byte PIN_BP = 2;
const byte PIN_POT = 0;
const int FREQ_MIN = 440;
const int FREQ_MAX = 2000;

void setup() {
  pinMode(PIN_BP,INPUT);
 }

void loop() {
  int valeur = analogRead(PIN_POT);
  valeur = map(valeur,0,1023,FREQ_MIN,FREQ_MAX);
  if (!digitalRead(PIN_BP)){
    tone(PIN_BUZZER,valeur,100);
  }
 }
```

## Comprendre :

**valeur = map(valeur,0,1023,FREQ_MIN,FREQ_MAX);**

// La fonction map() fait tous les calculs pour transformer une valeur qui varie entre 0 et 1023 (c'est une Conversion Analogique Numérique sur 10 bits) en un nombre compris entre FREQ_MIN et FREQ_MAX.

// La fonction map() peut aussi être utilisée avec des valeurs négatives, ainsi que pour inverser la pente de la courbe.

// Cette fonction nécessite pas mal de mémoire et prend du temps lors de l'exécution du programme

**tone(PIN_BUZZER,valeur,100);** // Le buzzer générera un son de fréquence valeur pendant 100 ms puis s'arrêtera, sauf si le BP est toujours appuyé. Cela permet de ne pas utiliser noTone().

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Faire varier la fréquence du buzzer entre 200 Hz et 1 kHz.

b) Utiliser un BP pour enregistrer la valeur du potentiomètre et un autre pour émettre le son.

c) Faire varier la fréquence du buzzer en sens inverse de la rotation du potentiomètre de 2 kHz à 100 Hz.

## Se souvenir :

map() ;

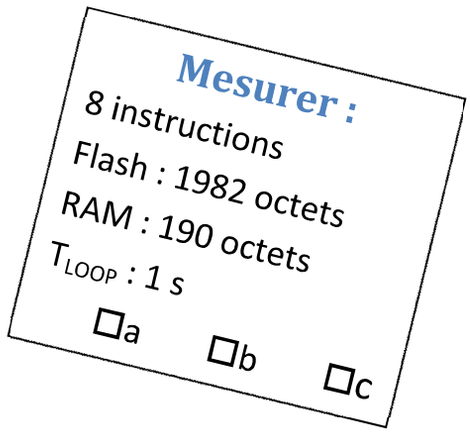
tone(broche, frequence, duree) ;

## Noter :

# Exercice 14

## Programmer :

**Tracer la courbe d'éclairement avec Regressi. La fréquence des mesures sera de 1 Hz.**




```
const byte PIN_LDR = 1;
int temps = 0;

void setup() {
  Serial.begin(9600);
 }

void loop() {
  int valeur = analogRead(PIN_LDR);
  Serial.print(temps++);
  Serial.print(",");
  Serial.println(valeur);
  delay(1000);
 }
```

.

## Comprendre :

**Serial.begin(9600);** // La liaison série est configurée avec un débit de 9600 bit/s. C'est le logiciel Regressi qui impose cette valeur.

// 9600 bit/s cela correspond environ à 1200 octet/s. En code ASCII un caractère est codé sur un octet. Sur un clavier, le record de vitesse de frappe maximale est de 1000 lettre/minute, donc 60 fois moins rapide.

// Les vitesses possibles pour communiquer avec un ordinateur sont : 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200.

**Serial.print(temps++);** // Cette instruction envoie la valeur de la variable temps par la liaison série puis ajoute 1 à temps.

// On aurait pu écrire ++temps : on aurait ajouté 1 à temps avant de l'envoyer.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) En utilisant map() ramener les valeurs de luminosité entre 0 et 100.

b) Changer la fréquence de mesure pour 5 Hz.

c) Faire varier la période de mesure avec le potentiomètre entre 100 ms et 10 000 ms.

## Se souvenir :

fonction(temps++);

fonction(++temps);

Regressi

## Noter :

# Exercice 15

## Programmer :

**Réaliser un thermostat. Allumer une LED si la température du LM35 est supérieure (ou inférieure) à un seuil défini.**

**Mesurer :**
9 instructions
Flash : 1022 octets
RAM : 15 octets
$T_{LOOP}$ : 152 µs
☐a ☐b ☐c

```
const byte PIN_LM35 = 2;
const byte PIN_ROUGE = 9;
const int SEUIL = 200;

void setup() {
  analogReference(INTERNAL);
  pinMode(PIN_ROUGE,OUTPUT);
 }

void loop() {
  int mesure = analogRead(PIN_LM35);
  mesure = map(mesure,0,1023,0,1100);
  if (mesure > SEUIL)digitalWrite(PIN_ROUGE, HIGH);
    else digitalWrite(PIN_ROUGE, LOW);
 }
```

## Comprendre :

**analogReference(INTERNAL) ;** // Le Convertisseur Analogique Numérique a besoin d'une tension de référence qui correspondra à la valeur maximale (1023 dans ce cas). Par défaut c'est la tension d'alimentation du µcontrôleur (environ 5 V). Mais on peut utiliser une tension de référence plus précise qui est fournie dans le µcontrôleur. Pour l'Arduino UNO la valeur de la tension de référence est de 1,1 V.

**LM35** // La tension fournie par le LM35 varie linéairement et vaut 0 V pour 0°C et 1 V pour 100°C. Le maximum que l'on puisse mesurer avec la référence interne est de 1,1 V , cela correspond à 110°C.

// Pour rester avec des nombres entiers on donne les températures x10, c'est-à-dire que 25,4°C sera affiché comme 254.

**if ( mesure > SEUIL ) digitalWrite (PIN_ROUGE, HIGH);**

// S'il n'y a qu'une seule instruction on peut se passer des accolades {}. Attention aux modifications !

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Envoyer la valeur de la mesure à l'ordinateur par la liaison série.

b) Générer un Beep quand le seuil est franchi (dans un sens ou dans l'autre).

c) Définir 2 seuils de température. La LED bleue est allumée quand il fait froid, la LED verte quand la température est entre les 2 seuils. La LED rouge quand il fait chaud.

## Se souvenir :

analogReference(INTERNAL);

LM35

## Noter :

# Exercice 16

## Programmer :

**Réaliser un thermostat. Allumer une LED si la température du LM35 est supérieure (ou inférieure) à un seuil réglable par le potentiomètre**

**Mesurer :**
17 instructions
Flash : 1290 octets
RAM : 15 octets
$T_{LOOP}$ : 20 ms
☐a ☐b ☐c

```
const byte PIN_LM35 = 2;
const byte PIN_POT = 0;
const byte PIN_ROUGE = 9;

void setup() {
  pinMode(PIN_ROUGE,OUTPUT);
 }

void loop() {
  int poubelle, mesure, seuil;
  analogReference(INTERNAL);
  poubelle = analogRead(PIN_LM35);
  delay(10);
  mesure = analogRead(PIN_LM35);
  mesure = map(mesure,0,1023,0,1100);
  analogReference(DEFAULT);
  poubelle = analogRead(PIN_POT);
  delay(10);
  seuil = analogRead(PIN_POT);
  seuil = map(seuil,0,1023,150,250);
  if (mesure > seuil)digitalWrite(PIN_ROUGE, HIGH);
    else digitalWrite(PIN_ROUGE, LOW);
 }
```

## Comprendre :

**int poubelle, mesure, seuil;** // Les trois variables sont déclarés en une seule fois et ne sont pas initialisées.

// On aurait pu initialiser seuil en écrivant seuil = 200.

// Elles ne sont utilisables qu'à l'intérieur de la fonction loop().

**analogReference(INTERNAL);**
**poubelle = analogRead(PIN_LM35);**
**delay(10);**
**mesure = analogRead(PIN_LM35);**

// On change de tension de référence car le potentiomètre fournit une tension entre 0 V et 5 V, supérieure à 1,1 V.

// La documentation du µcontrôleur (et l'aide Reference d'Arduino) indique que lorsque l'on change de tension de référence la conversion suivante est mauvaise et qu'il faut attendre que la tension se stabilise.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Envoyer la valeur de la mesure et la valeur du potentiomètre à l'ordinateur par la liaison série.

b) Générer un Beep quand le seuil est franchi (dans un sens ou dans l'autre).

c) Définir 2 seuils de température séparés de 1°C, le potentiomètre permet de faire varier ces seuils. La LED bleue est allumée quand il fait froid, la LED verte quand la température est entre les 2 seuils. La LED rouge quand il fait chaud.

## Se souvenir :

int poubelle = 0, mesure, seuil = 200;

analogReference() : le monde réel est parfois dur à comprendre ☹!

## Noter :

# Exercice 17

## Programmer :

**Réaliser un variateur de lumière réglé par le potentiomètre.**

**Mesurer :**

6 instructions

Flash : 948 octets

RAM : 9 octets

$T_{LOOP}$ : 112 µs

☐a ☐b ☐c

```
const byte PIN_POT = 0;
const byte PIN_LED = 11;

void setup() {
  pinMode(PIN_LED,OUTPUT);
 }

void loop() {
  int mesure = analogRead(PIN_POT);
  mesure = mesure/4;
  analogWrite(PIN_LED,mesure);
 }
```

## Comprendre :

**const byte PIN_LED = 11;** // Toutes les broches ne peuvent pas être utilisées pour la variation. Sur l'Arduino il y a un symbole ~ pour indiquer cette possibilité.

// La variation se fait par MLI (Modulation par Largeur d'Impulsion).

**analogWrite(PIN_LED,mesure);**

// La tension moyenne sur la broche vaut 0 V quand mesure vaut 0 et vaut 5 V (le maximum) quand mesure vaut 255.

**mesure = mesure/4;** // mesure varie entre 0 et 1023, en divisant par 4 on se ramène de 0 à 255. C'est plus simple que d'utiliser la fonction map().

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Faire varier la lumière de 2 LED simultanément de la même façon.

b) Faire varier la lumière de 2 LED simultanément en inverse (quand l'une s'éclaire l'autre s'éteint).

c) Faire varier le niveau d'éclairage d'une LED rouge avec la luminosité ambiante de façon à ne pas être ébloui dans l'obscurité et de la voir en plein jour.

## Se souvenir :

analogWrite(pin,valeur);

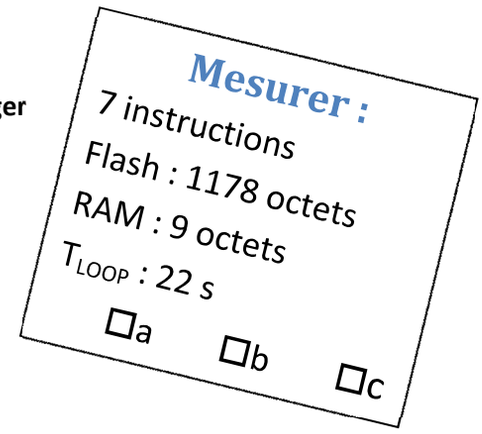
mesure = mesure/4

## Noter :

# Exercice 18

## Programmer :

**Générer des couleurs avec une LED RGB. Les couleurs vont changer toutes les 0,1s.**




```
const int TEMPO = 100;
byte led[] = {9,10,11};

void setup() {
  for (byte i=0;i<3;i++){
    pinMode(led[i],OUTPUT);
  }
 }

void loop() {
  for (int r=0;r<256;r=r+51){
    analogWrite(led[0],r);
    for (int g=0;g<256;g=g+51){
      analogWrite(led[1],g);
      for (int b=0;b<256;b=b+51){
        analogWrite(led[2],b);
        delay(TEMPO);
        }
      }
    }
  }
```

## Comprendre :

**for (int r=0;r<256;r=r+51){ }** // Cette boucle for s'effectue 6 fois : r =0 puis 51, 102,153, 204 et 255. Le pas (51) a été choisi pour couvrir toute la gamme d'éclairage de la LED.

```
for (int r=0;r<256;r=r+51){
  analogWrite(led[0],r);
  for (int g=0;g<256;g=g+51){
   analogWrite(led[1],g);
```

// Les 3 boucles for sont imbriquées les unes dans les autres. Celle qui s'effectue le plus souvent est celle qui est la plus à l'intérieur. Il faut respecter une indentation croissante des lignes de code.

// On obtient donc 6x6x6 = 216 couleurs différentes.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Faire varier la temporisation avec le potentiomètre (entre 10 ms et 500ms).

b) Faire varier le nombre de couleurs avec le potentiomètre.

c) Changer le sens de variation d'une ou plusieurs boucles.

## Se souvenir :

for (int r=0;r<256;r=r+51){ }

Indentation

## Noter :

# Exercice 19

## Programmer :

**Programmer la couleur d'une LED RGB avec 2 BP et un potentiomètre. Un BP permet de choisir la couleur à régler, l'autre BP recopie la valeur du potentiomètre**

**Mesurer :**
13 instructions
Flash : 1514 octets
RAM : 15 octets
$T_{LOOP}$ : 10 µs (0,2 s)
☐a ☐b ☐c

```
const byte BP_CHOIX = 2;
const byte BP_REGLAGE = 3;
const byte PIN_POT = 0;
byte led[] = {9,10,11};
byte memoire = 0, index = 0;

void setup() {
  for (byte i=0;i<3;i++){
    pinMode(led[i],OUTPUT);
  }
  pinMode(BP_CHOIX,INPUT);
  pinMode(BP_REGLAGE,INPUT);
 }

void loop() {
  if (!digitalRead(BP_CHOIX)){
    if (memoire==0) {
      index = (index+1)%3;
       memoire = 1;
       delay(200);
    }
  }
  else memoire =0;
  if (!digitalRead(BP_REGLAGE)){
    analogWrite(led[index],analogRead(PIN_POT)/4);
    }
}
```

## Comprendre :

**index = (index+1)%3;** // On ajoute 1 à index puis si index est égal à 3 index est mis à 0.

// Ce petit calcul mathématique avec la fonction MODULO (%) est ce qui a de plus efficace pour parcourir toutes les valeurs entre 0 et 2 en boucle.

```
if (!digitalRead(BP_CHOIX)){
  if (memoire==0) {
    action;
    memoire = 1;
    delay(200);
  }
}
else memoire =0;
```

// On réutilise la partie de code du télérupteur (Exercice 6) qui gère le BP.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Générer un beep quand on change de couleur de base.

b) Faire la même chose avec seulement un BP.

c) Envoyer les valeurs des couleurs de base à l'ordinateur à intervalle régulier.

## Se souvenir :

index = (index+1)%3;

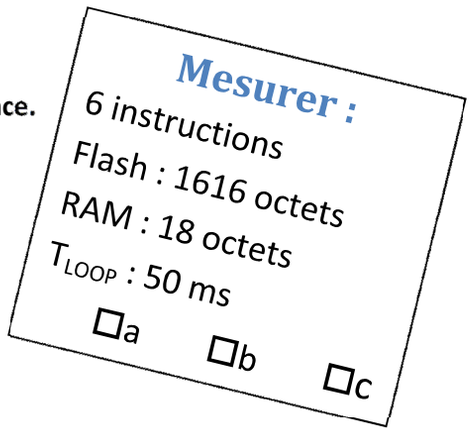
Réutiliser le code qui marche !

## Noter :

# Exercice 20

## Programmer :

**Faire varier la couleur d'une LED RGB aléatoirement en permanence.**




```
byte led[] = {9,10,11};
byte index=0;

void setup() {
  for (byte i=0;i<3;i++){
    pinMode(led[i],OUTPUT);
  }
 }

void loop() {
  analogWrite(led[index],random(100));
  index = (index + 1)%3;
  delay(50);
}
```

## Comprendre :

**index = (index+1)%3;** // On ajoute 1 à index puis si index est égal à 3 index est mis à 0.

// Ce petit calcul mathématique avec la fonction MODULO (%) est ce qui a de plus efficace pour parcourir toutes les valeurs entre 0 et 2 en boucle.

// La boucle est réalisée par la fonction loop().

**random(MAX);** // La fonction renvoie un nombre presque au hasard entre 0 et MAX-1. Chaque fois qu'on fait un appel à cette fonction elle renvoie un nombre différent.

// La fonction random() s'utilise aussi avec random(MIN,MAX)

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Faire varier la vitesse de changement avec le potentiomètre.

b) Adapter le niveau d'éclairage à la luminosité ambiante.

c) Modifier progressivement la couleur.

## Se souvenir :

random();

Réutiliser le code qui marche !

## Noter :

# Exercice 21

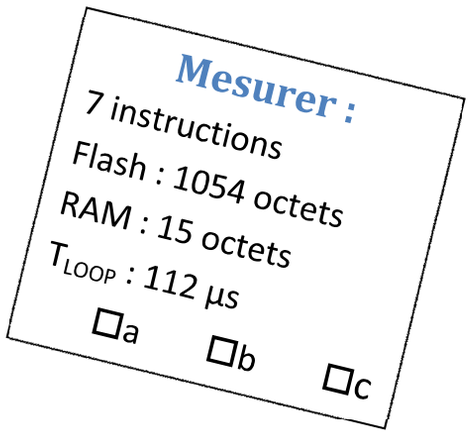
## Programmer :

**Faire un variateur de lumière blanche.**

```
const int PIN_POT = 0;
byte led[] = {9,10,11};
byte valeur = 0, valeurAvant = 0;

void setup() {
  for (byte i=0;i<3;i++){
    pinMode(led[i],OUTPUT);
  }
 }

void loop() {
  valeur = analogRead(PIN_POT)/4;
  if (valeur != valeurAvant){
    valeurAvant = valeur;
    for (byte i=0;i<3;i++){
      analogWrite(led[i],valeur);
    }
  }
}
```




## Comprendre :

**for (byte i=0;i<3;i++){**
**analogWrite(led[i],valeur);**
**}** // Pour émettre une lumière blanche on éclaire les 3 LED avec la même intensité.

**!=** // Si la comparaison d'égalité s'écrit ==, le fait d'être différent s'écrit !=.

**if (valeur != valeurAvant){**
**valeurAvant = valeur;**
**action;**
**}** // On n'effectue l'action que si la valeur du potentiomètre a changée.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Inverser le sens du fonctionnement du potentiomètre.

b) Changer la couleur de la lumière pour avoir une couleur plus chaude (avec du jaune).

c) Générer un Beep quand on tourne le potentiomètre, en plus de modifier l'intensité de la lumière.

## Se souvenir :

!=

if (valeur != valeurAvant){ valeurAvant = valeur ; action ;}

## Noter :

# Exercice 22

## Programmer :

**Changer l'état d'une sortie à chaque appel de la fonction loop() pour pouvoir mesurer la durée $T_{LOOP}$.**

```
const int PIN_MESURE = 7;
byte sortie = 0;

void setup() {
  pinMode(PIN_MESURE,OUTPUT);
//  DDRD = 0b10000000;
 }

void loop() {
  digitalWrite(PIN_MESURE,sortie);
  sortie = !sortie;
//  PORTD = PORTD ^ 0b10000000;
}
```

**Mesurer :**

5(4) instructions

Flash : 754(454) octets

RAM : 10(9) octets

$T_{LOOP}$ : 3,52(0,44) µs

☐a ☐b ☐c

## Comprendre :

sortie = !sortie; // On change la valeur de sortie à chaque appel de la fonction loop(). Quand on mesure la fréquence sur la broche 7, la période associée (T=1/f) correspond à une durée haute et une durée basse dons $2xT_{LOOP}$.

// Rappel : la fonction setup() est appelée une fois au démarrage (mise sous tension ou appui du bouton Reset), la fonction loop() est appelée en permanence.

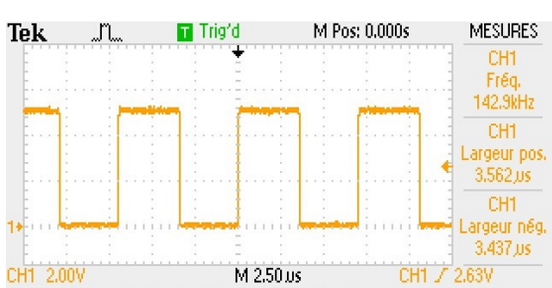
// On mesure la fréquence avec un oscilloscope ou un fréquencemètre. Attention aux multimètres leur plage de mesure peut être insuffisante. (Exemple le MX24 mesure les fréquences jusqu'à 500 kHz).

**PORTD = PORTD ^ 0b10000000;**

// Le PORTD est un registre du µcontrôleur (une case mémoire d'un octet) qui est copié sur certaines broches. Le bit 7 de PORTD correspond à D7.

// L'opération réalisée est un OU EXCLUSIF bit à bit qui sera vue plus tard.

**Résultats des mesures**




// f = 142 kHz, $T_{LOOP}$ = 1/(2*f) = 3,52 µs pour les instructions non commentées.

// f=1,14 MHz, Tloop = 1/(2*f) = 0,44 µs si on utilise les instructions commentées.

// La deuxième méthode est presque 10 fois plus rapide !

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Mesurer $T_{LOOP}$ pour l'exercice 3 avec digitalWrite().

b) Mesurer $T_{LOOP}$ pour l'exercice 12 avec digitalWrite(). Comparer le résultat avec et sans appui du BP.

c) Refaire l'exercice b) mais en utilisant DDRD et PORTD.

## Se souvenir :

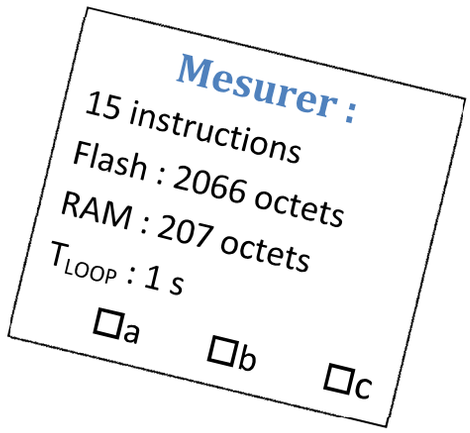
Mesure de la vitesse d'exécution d'un programme

## Noter :

# Exercice 23

## Programmer :

**Mesurer la luminosité toutes les secondes et faire une moyenne sur 8 mesures. Envoyer le résultat à l'ordinateur**




```
const byte PIN_LDR = 1;
int lum[8];
byte index = 0;
int moyenne = 0;

void setup() {
  Serial.begin(115200);
 }

void loop() {
  index%=8;
  lum[index]=analogRead(PIN_LDR);
  moyenne = 0;
  for (int i =0;i<8;i++){
    moyenne = moyenne + lum[i];
  }
  moyenne = moyenne/8;
  Serial.print(lum[index]);
  Serial.print(",");
  Serial.println(moyenne);
  delay(1000);
  index++;
 }
```

## Comprendre :

**moyenne = 0** // Il faut remettre à 0 la variable avant le calcul de la moyenne. Puis on fait la somme de toutes les valeurs et on divise par 8. On utilise une seule variable.

**lum[index]** // la taille du tableau est 8, index peut prendre les valeurs de 0 à 7.

**index%=8** // C'est une façon rapide d'écrire index = index%8. On s'assure ainsi que index ne dépasse pas 7.

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Utiliser la valeur moyenne pour contrôler le niveau d'éclairement d'un éclairage blanc.

b) Utiliser une constante pour le nombre d'échantillon. Tester pour 16 et 32 échantillons.

c) Déterminer le minimum, le maximum et envoyer toutes les secondes : min, moy, max.

## Se souvenir :

%=

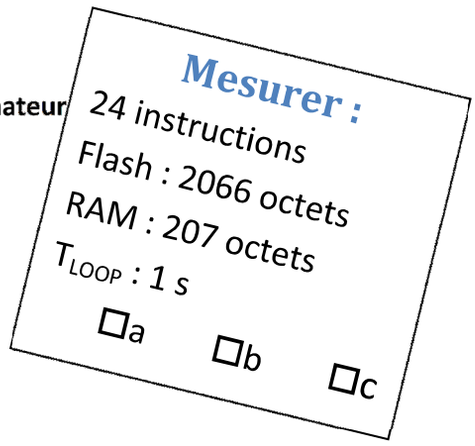
Faire une moyenne

## Noter :

# Exercice 24

## Programmer :

**Faire une horloge et envoyer l'heure toutes les secondes à l'ordinateur**
**On choisit la grandeur à régler avec le potentiomètre et on l'augmente avec un BP.**




```
const byte PIN_POT = 0;
const byte PIN_BP = 2;
byte led[] = {9,10,11};
byte t[] ={0,0,0};
byte choixReglage = 0;
byte bp = 0, bpAvant = 0;

void setup() {
  for (int i=0;i<3;i++){
    pinMode(led[i],OUTPUT);
  }
  pinMode(PIN_BP,INPUT);
  Serial.begin(115200);
 }

void loop() {
  bp = !digitalRead(PIN_BP);
  choixReglage = (analogRead(PIN_POT)/350)%3;
  for (int i=0;i<3;i++){
    if (i==choixReglage) digitalWrite(led[i],HIGH);
    else digitalWrite(led[i],LOW);
  }
  if (bp==1){
    if (bpAvant==0)t[choixReglage]++;
  }
  bpAvant = bp;
  t[0] = (t[0]+1)%60;
  if (t[0]==0) {
    t[1] = (t[1]+1)%60;
    if (t[1]==0) t[2] = (t[2]+1)%12;
  }
  Serial.print(t[2]); Serial.print(":");
  Serial.print(t[1]); Serial.print(":");
  Serial.println(t[0]);
  delay(1000);
 }
```

## Comprendre :

**byte t[ ]** // Ce tableau contient les valeurs des secondes, minutes et heures. t[0] correspond aux secondes, t[1] correspond aux minutes et t[2] correspond aux heures.

**t[0] = (t[0]+1)%60;** // la fonction MODULO (%) est très pratique pour gérer le temps. En effet toutes les 60 secondes il faut remettre à zéro les secondes et incrémenter les minutes. Là on incrémente les secondes puis on prend le reste de la division par 60. Si on a 0 c'est que c'est une nouvelle minute. On fait de même pour les minutes, puis les heures.

**Organisation du programme** // Il y a 5 parties :

// 1°) La lecture des entrées.

// 2°) La détermination du choix du réglage

// 3°) La gestion du BP et le réglage

// 4°) Le calcul du temps courant

// 5°) L'envoi par la liaison série

## Faire marcher :

- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Générer un Beep à chaque minute et à chaque heure (accélérer l'horloge pour tester).

b) Faire un chronomètre avec un bouton Start et un bouton Stop.

c) Améliorer la réactivité du programme en divisant le $T_{LOOP}$ par 10.

## Se souvenir :

Utiliser des tableaux

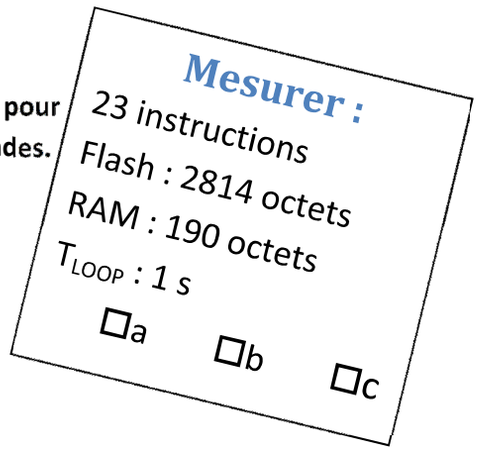
%

## Noter :

# Exercice 25

## Programmer :

**Thermomètre lumineux. Rouge pour 32°C, vert pour 25°C et bleu pour 18°C. La température est envoyée à l'ordinateur toutes les secondes.**




```
const byte PIN_LM35 = 2;
const byte ROUGE = 9, VERT = 10, BLEU = 11;
const int MILIEU = 250;   //25°C
const int PERIODE = 1000;

void setup() {
  Serial.begin(9600);
  analogReference(INTERNAL);
 }

void loop() {
  int bleu = 0, vert = 0, rouge = 0;
  int mesure = analogRead(PIN_LM35);
  mesure = map(mesure,0,1023,0,1100);
  Serial.print(mesure/10); Serial.print(',');
  Serial.println(mesure%10);

  bleu = 3*(MILIEU - mesure);
  if (bleu<=0)  bleu = 0;
  if (bleu>=255) bleu = 255;
  rouge = 3*(mesure - MILIEU);
  if (rouge<=0)  rouge = 0;
  if (rouge>=255)  rouge = 255;
  vert = 255 - (rouge + bleu);

  analogWrite(ROUGE,rouge);
  analogWrite(VERT,vert);
  analogWrite(BLEU,bleu);

  delay(PERIODE);
 }
```

## Comprendre :

**‘,’** // Le guillemet simple s'utilise uniquement avec les caractères uniques. Si on voulait envoyer plusieurs caractères (une chaîne de caractères) il faudrait utiliser les doubles guillemets, par exemple "Celsius".

**Organisation du programme** // Il y a 3 parties :

// 1°) La lecture des entrées.

// 2°) Les calculs

// 3°) La gestion des sorties (LEDs et liaison série)

**Serial.print(mesure/10); Serial.print(',');**
**Serial.println(mesure%10);** // On détermine la partie entière et la partie décimale de la température en 2 opérations : la division euclidienne par 10 et l'opération modulo (%) 10.

// Une autre méthode consiste à utiliser une variable de type float

## Faire marcher :

- ☐ Utiliser l'icône ✓ (Vérifier) pour chercher des erreurs.
- ☐ Ne pas oublier les ; à la fin de chaque instruction !
- ☐ Faire attention aux minuscules/MAJUSCULES.
- ☐ Vérifier l'exactitude du programme avec la coloration syntaxique.
- ☐ Les constantes doivent être écrites en MAJUSCULES et systématiquement initialisées.
- ☐ Vérifier que les indices ne dépassent pas la taille des tableaux et qu'on a les bons [] () {}.
- ☐ Vérifier que les opérateurs de comparaison sont corrects.
- ☐ Vérifier que la vitesse du moniteur série est la bonne.
- ☐ Avant de téléverser le programme avec l'icône ➔ , vérifier que la carte et le port de communication sélectionnés sont les bons ( Outils – Type de carte et Port).

## S'entrainer :

a) Faire la même chose avec le potentiomètre.

b) Faire varier la couleur de la LED RGB en fonction de la luminosité.

c) Générer un Beep (et un seul !) lorsqu'une température précise est atteinte.

## Se souvenir :

Organisation d'un programme

Nombre décimal

## Noter :