**COURS ARDUINO**

[**https://openclassrooms.com/fr/courses/2778161-programmez-vos-premiers-montages-avec-arduino**](https://openclassrooms.com/fr/courses/2778161-programmez-vos-premiers-montages-avec-arduino)

référence : <https://www.arduino.cc/reference/en/> (recherche google directe # processing)

**1\_ DEFINITIONS ARDUINO** (société italienne Smart Projects)

* Arduino est un **circuit imprimé** en **matériel libre**  **photo1\_1**
* sur lequel se trouve un **microcontrôleur** qui peut être programmé
* pour analyser et produire des **signaux électriques**.
* **circuit imprimé :** plaque d’isolant sur laquelle :
* est gravé un circuit électrique qui permet de relier en eux
* plusieurs composants électroniques reliés entre eux.
* **matériel libre :**plans de la carte elle-même sont accessibles par tout le monde, gratuitement.
* **microcontrôleur :** c’est lui qu’on programme **photo1\_2**
* cœur de la carte # ordinateur (ROM, RAM 32ko sur UNO !, processeur et entrées/sorties)
* **signaux électriques :**capable de produire ou de capter ces signaux selon la programmation.
* **Schéma général = principales interactions en jeu :**
* se connecte sur l’ordinateur pour être programmée / peut ensuite fonctionner seule si elle est alimentée
* permet de recevoir/transmettre des informations depuis ou vers des matériels électroniques :

diodes, potentiomètres, récepteurs, servo-moteurs, moteurs, détecteurs...

**photo1\_3 :** vert = électricité / orange : téléversement / bleu : réel

**1\_ MATERIEL / LOGICIEL**

#### Matériel :

#### carte Arduino / breadboard (plaque d’essai) / fils de connexion (jumpers) photo1\_4 / 1\_5

#### composants que vous souhaitez contrôler avec votre carte Arduino : photo1\_6

#### ****LED**** (pour produire de la lumière)

* **Résistances** (pour contrôler le courant) : 5 de 220 Ω, 2 de 10 kΩ, 2 de 1 KΩ.
* **Contacteurs** ou **bouton poussoir** (pour interagir avec le montage)
* **Potentiomètres** (pour créer des variations) : 1 de 10 KΩ
* **Servo-moteurs** (pour créer des mouvements) : 1 suffira, pilotable sur 5V.
* **Moteur CC** (courant continu) : 2, qui fonctionnent avec une tension entre 3V et 6V.
* Un transistor bipolaire NPN (PN2222) et un MOSFET (IRF540N) pour piloter les moteurs.
* Une diode Schottky (1N4001) qui autorise le passage du courant dans un seul sens.
* Une puce L293D, pour piloter les moteur CC.
* etc.

#### Logiciels :

#### Logiciel : inferface de programmation photo1\_7 <https://www.arduino.cc/> (version française dispo)

* [Fritzing](http://fritzing.org/home/) : utilitaire permet schématiser les montages **photo1\_8** <https://fritzing.org/>

**2\_ PRINCIPES PROGRAMMATION**

#### structure de base : setup(), loop() photo1\_7

#### interface de l’éditeur : photo1\_7 5 boutons :

#### V : vérification syntaxique (rem : interprété versus compilé) photo2\_1 / 2-2

#### 🡪 : téléversement : transfert du programme de l’ordinateur à la carte arduino réliée par cable USB

#### = une fois en mémoire, le programme est exécuté tant qu’il est alimenté électriquement

#### ↓ : enregistrement sketch (fichier .ino)

#### ↑ : ouvrir sketch (ouvre les exemples)

#### leds carte : photo2\_3 :

* 3 petits rectangles notés L, TX et RX, et un autre à droite noté ON = “**L**ight-**E**mitting **D**iodes”
* LED **ON** = allumée sous tension
* LEDs **TX** et **RX =** clignotent envoi/réception d’informations
* LED **L** = clignote si on appuie sur le bouton reset

= reliée à la connexion 13 donc s’allume si on envoie du courant dessus

#### Programme led 13 : photo2\_4 :

1. Dire que la connexion 13 puisse envoyer du courant (et pas en recevoir).
2. Dire d’envoyer du courant dans la connexion (= **pin**) 13

Remarques :

* + impossible à arrêter éteindre … même avec bouton reset.
  + Mots clés du langage : **HIGH** vaut 1=+5V, **LOW** vaut 0=0V

#### Programme blink blink : photo2\_5 / sketch\_2\_0\_blinkblink.ino

|  |  |
| --- | --- |
| 1. dire que la connexion 13 doit pouvoir envoyer du courant 2. envoyer du courant en connexion 13 (la diode s'allume) 3. arrêter d'envoyer du courant en connexion 13 (la diode s'éteint) 4. recommencer au point numéro 2 à l'infini. | void setup(){  pinMode(13,OUTPUT);  }  void loop(){  digitalWrite(13,HIGH);  digitalWrite(13,LOW);  } |

#### Fréquence et période :

Doc « *Chaque module possède au moins un régulateur linéaire 5 V et un oscillateur à quartz 16 MHz »*

= tension de sortie au niveau des connexions = 5V / processeur à 16 MHz

16 000 000 d'actions/s <> 16 000 000 d'instructions/s

= une action machine = passage ou non d'un courant électrique (1 et 0)

= une instruction nécessite une multitude d'actions machine pour se réaliser

| Exemple | Fréquence | Action |
| --- | --- | --- |
| Respiration | 0,5 Hz | 1 respiration toutes les 2 secondes |
| Seconde | 1 Hz | 1 seconde toutes les secondes |
| Pouls humain | 1,2 Hz | 1,2 battement par seconde soit 72 par minute environ |
| Courant alternatif | 50 Hz | 50 oscillations par seconde |
| La du diapason | 440 Hz | 440 oscillations par seconde |
| Ondes FM | entre 23 et 95 KHz | entre 23 000 et 95 000 vibrations par seconde |
| Arduino | 16 MHz | soit 16 000 000 d'actions machine par seconde |
| Processeur actuel | 3 Ghz | soit 3 000 000 000 d'actions machine par seconde |

* **la LED clignote à un peu moins de 77 KHz =** 77 000 fois j'allume/j'éteins par seconde <> œil 25Hz !

= principe du calcul : différence de temps avec l’horloge interne avant après

|  |
| --- |
| void setup() {  Serial.begin(9600);  }  void loop() {  unsigned long tempsFin; //variable pour noter le temps final  unsigned long tempsDepart=micros(); //variable du temps de départ initialisée  //bloc à tester  tempsFin=micros(); //on récupère à nouveau le temps actuel  Serial.println(tempsFin-tempsDepart); // différence en microsecondes  } |

* **delay()** = instruction d’attente en ms : ajouter : delay(1000); **photo2\_6**

**3\_ LANGAGE DE PROGRAMMATION = constantes / variables / test**

#### constantes = mot du langage à valeur fixée par le langage ou utilisateur (exp : LOW ,  HIGH ,  OUTPUT)

#### intérêt : 1/ paramétrage 2/ nom signifiant sketch\_3\_1\_blinkblink\_delay.ino

|  |
| --- |
| const int CONNEXION=13;  void setup() {  Serial.begin(9600);  pinMode(CONNEXION,OUTPUT);  }  void loop() {  digitalWrite(CONNEXION,HIGH);  delay(1000);  digitalWrite(CONNEXION,LOW);  delay(1000);  } |

#### variables

* déclaration en 2 temps (moins couteuse en mémoire)

1/réserve une place en mémoire en la nommant (1 fois seulement)

2/ affecte une valeur à la variable (possible x fois).

* typage des variables et constantes = OBLIGATOIRE

Explication : **photo3\_1**

* + mémoire d'un processeur = ensemble d'espaces de même taille (1 byte) qui « se suivent »
  + on peut donc les retrouver grâce à leur position sur cette ligne virtuelle
  + leur taille est d'un **byte=**8 bits= 0 ou 1
  + à l’appel d’une variable, le processeur va cherche sur la ligne = **adresse** et récupére le contenu
  + **le type permet de récupérer le quantité de cases approprié**

| type | taille en byte | valeurs stockées |
| --- | --- | --- |
| **boolean** | 1 | true ou false |
| **char** | 1 | un caractère ou un entier entre -128 et 127 |
| **unsigned char** | 1 | un entier entre 0 et 255 |
| **byte** | 1 | un entier entre 0 et 255 |
| **int** | 2 | un entier entre -32768 et 32767 (256x256) |
| **unsigned int** | 2 | un entier entre 0 et 65535 |
| **float** | 4 | un décimal, précis à 7 chiffres après la virgule |

Remarque : conventions d’écriture : CONSTANTES / nomDeVariable au noms signifiants.

Pas d’accent, ni espace, ni caractères spéciaux et ne peuvent commencer par un chiffre

* portée des variables (ou « scope »)
  + variable globale : en tête de programme avant le setup() (l’espace mémoire n’est jamais libérée)
  + variable locale : entre {} = portée limitée au bloc de code défini entre les {}

(une fois sorti du bloc, l’espace mémoire de la variable est libérée)

#### moniteur série : fenêtre accessible en cliquant sur la loupe du logiciel Arduino

#### remarque : console = moyen simple d'afficher des infos provenant de votre Arduino qui n’a pas d’écran

* reçoit et envoie des infos sous forme de séries de bytes aux ports concernés

= Arduino peut donc envoyer des infos à l'ordinateur, les afficher, les traiter en temps réel.

* Mise en œuvre : bibliothèque/library **Serial sketch\_3-2\_setup-loop.ino**

*déf :* ***bibliothèque*** *= fichier contenant des sous-programmes, des variables et des constantes.*

<https://www.arduino.cc/reference/en/language/functions/communication/serial/>

|  |
| --- |
| void setup(){  Serial.begin(9600);// initialisation de la communication  Serial.println("Communication initialisée");// envoi d'un message  }  void loop(){  Serial.println("Je suis dans la boucle !");//envoi d'un autre message  } |

Remarque : 9600 = nombre de caractères (8bits) par seconde (bauds <> bps = bits/s)

* Exécution : diode TX de votre Arduino est allumée = transmission envoyée par l'Arduino
* Accès / affichage : icône Loupe

= défilement des informations en temps réel / ajouter delay(2000); pour un envoi toutes les 2 secondes

* Compléments
* utilise les connexions 0 et 1 (0 pour diode RX / 1 pour diode TX) = ne pas les utiliser
* affichage des caractères accentués ou spéciaux n'est pas pris en compte
* vitesse de communication : 9600 bauds (defaut) mais aussi 4800/9600/19200/38400

<https://www.arduino.cc/reference/en/language/functions/communication/serial/begin/>

* bouton reset n'effacera pas les messages précédents affichés sur le moniteur. Relancer = loupe.

#### Programme « Hello Arduino World ! » sketch\_3-3\_helloword.ino

|  |
| --- |
| void setup() {  Serial.begin(9600); //initialisation de la communication  Serial.println("Hello"); //affichage du texte  Serial.println("Arduino");  Serial.println("World !");  }  void loop() { } |

#### remarque : Serial.println() : retour à la ligne / Serial.print() : sans retour à la ligne

#### condition if = test qui renvoie vrai (true) ou faux (false) sketch\_3-4\_condition\_if.ino

|  |
| --- |
| int affichageFait;// déclaration de la variable  void setup(){  Serial.begin(9600);  affichageFait=0;//initialisation de la variable  }  void loop(){  if (affichageFait==0){ // exécuté que si la condition est vérifiée  Serial.println("Hello");  Serial.println("Arduino");  Serial.println("World !");  affichageFait=1;//on passe la variable à 1 pour ne plus exécuter le code  }  } |

#### Remarque : conditions : == / >= / <= / > / < / !=

**4\_ LANGAGE DE PROGRAMMATION = boucles / calculs**

* **programme "Arduino compte seul" :** compte jusqu'à 20 (2Hz) et fait clignoter LED 13 = nb de fois sa valeur

|  |  |
| --- | --- |
| Étape1 : compteur | Étape 2 : test |
| int compteur;  void setup() {  Serial.begin(9600);  compteur=1;  }  void loop() {  Serial.println(compteur);  compteur=compteur+1;  } | int compteur;  void setup(){  Serial.begin(9600);  compteur=1;  }  void loop(){  if (compteur<=20){  Serial.println(compteur);  compteur=compteur+1;  }  } |

Remarque : int sur 2 bytes donc max 32768

* boucle for : exécute plusieurs fois un bloc d’instructions

|  |  |
| --- | --- |
| for(**1, 2, 3**)  **1 =** **initialisation** de la variable  **2 = condition à tester** pour continuer ou non  **3 = action sur le compteur** (in-dé-crémentation) | for (int t=0;t<10;t=t+1) {  code à exécuter  } |

* type boolean : vrai/faux (true = 1 ou false = 0)

remarques : (etat==true) # (etat) **NÉCESSITE INDENTER**

|  |  |
| --- | --- |
| Etape 3 : for | Étape4 : boolean |
| int cpt;  void setup() {  Serial.begin(9600);  }  void loop() {  for (int cpt=0; cpt<=20; cpt++){  Serial.println(compteur);  }  } | boolean affichage; //variable pour stopper  void setup(){  Serial.begin(9600);  affichage=true; //initialisation à true  Serial.println("\*\*\* Debut programme \*\*\*");  }  void loop(){  if (affichage){// test si affichage true  for (int cpt=0;cpt<=20;cpt++){  Serial.println(compteur);  }  affichage=false; // on passe si false  }  } |

* programme final **sketch\_4-1\_boucle\_for.ino**

|  |
| --- |
| boolean affichage; //variable pour stopper l'affichage  int numPin;  void setup() {  numPin=13;  pinMode(numPin,OUTPUT);  Serial.begin(9600);  affichage=true; //initialisation de la variable à true  Serial.println("\*\*\* Debut du programme \*\*\*");  }  void loop() {  if (affichage) { // test si affichage vaut true  //boucle de comptage  for (int compteur=1;compteur<=20;compteur=compteur+1) {  Serial.println(compteur);    //boucle de clignotement : compteur sert de limite à la boucle  for (int nbClignote=0;nbClignote<compteur;nbClignote=nbClignote+1) {  digitalWrite(numPin,HIGH); //allume  delay(250);  digitalWrite(numPin,LOW); //eteind  delay(250);  }  delay(1000); //attente de 1s  }  affichage=false; // on passe affichage à false  Serial.println("\*\*\* fini ! \*\*\*");  }  } |

remarques :

* + 3 façon d’incrémenter une variable : variable=variable+1; / variable+=1; variable ++;
  + Comme un seul passage, on peut tout faire dans setup() et supprimer variable affichage.
* **programme " Table de multiplication " :** table de multiplication par 7 des nombres de 0 à 14
  + contraintes : un seul affichage / 7 en paramètre / résultat = **photo4\_1**
  + remarque : saut de ligne = Serial.println()

**sketch\_4-2\_table\_multiplication.ino**

|  |
| --- |
| int numTable; // variable pour la table concernée  boolean affiche; // variable d'affichage  void setup() {  affiche = true; // initialisation à true  numTable = 7; // iniialisation à 7  Serial.begin(9600); //initialisation de l'affichage    //Affichage de l'entête du programme  Serial.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  Serial.println("Table de multiplication");  Serial.print("La table de : "); //pas de retour à la ligne  Serial.println(numTable); // affichage de la variable  Serial.println(); // saut de ligne pour aérer  }  void loop() {  if (affiche) { // test si vrai  // boucle de progression pour la multiplication  for (int t = 0; t < 15; t++) {  int resultat = numTable \* t; // variable pour stocker le résultat    // Affichage de la ligne  Serial.print(t);  Serial.print(" x ");  Serial.print(numTable);  Serial.print(" = ");  Serial.println(resultat); // ou Serial.println(numTable\*t); pour ne pas créer résultat  }  Serial.println(); // saut de ligne  Serial.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  affiche=false; // passage à false pour ne plus afficher  }  } |

**5\_ JEUX DE LUMIERES = 1 LED**

* **Notions d’électricité =** un déplacement de particules chargées dans un conducteur = courant

analogie # rivière qui coule source à embouchure, puis à l'embouchure, pompe qui ramène l'eau à la source

* + la charge électrique (+ ou -) est une propriété de la matière
  + 2 charges identiques se repoussent et 2 charges contraires s'attirent
* tension ( en volt / symbole V) = **différence de potentiel** crée par le courant électrique

Électricité (différence entre charges + et -) # rivière hauteur source>embouchure

Tension Arduino UNO entre bornes (pin 0 à 13) = +5 V (en courant continu)

| Exemple | Tension |
| --- | --- |
| Pile | 1,5 V ; 4,5 V ; 9 V |
| Batterie voiture | 12 V ; 24 V |
| Prise de courant | 110 V ; 220 V |
| Ligne à haute tension | 10 000 V ; 500 000 V |
| Foudre | 1 000 000 V |

Alimentation de l’Arduino :

* **câble USB** : ordinateur fournit un courant de +5 V
* **pile 9 V** (entre 7 V et 12 V) (prise ronde en respectant + et -) : Arduino transforme les 9V en 5V
* **transformateur** (220V alternatif->9V continu) (prise ronde)(vérifier 9V sinon endommage l’Arduino)

Circuit électrique :

*dipôle = composant électrique a 2 bornes (ampoules, moteurs, piles, interrupteurs...)*

générateur = **dipôle actif**#pompe, ampoule = **dipôle passif**#cascade

🡪 **photo5\_1** : différence de potentiel Ug = Uc / **photo5\_2** : *Ug*=*Ud*1+*Ud*2

*lois de Kirchhoff* (applicable en circuit fermé et courant continu)

**= conservation des charges (loi des mailles) = *Ug*=*Ud*1+*Ud*2**

<https://fr.wikipedia.org/wiki/Lois_de_Kirchhoff>

* intensité ( en ampère / symbole A) = **quantité de courant** qui passe dans un endroit donné du circuit

nombre de particules/s # débit de la rivière à cet endroit (si la rivière se sépare en 2, débit se répartit)

*loi de Kirchhoff* **= conservation du débit (loi des nœuds)** 🡪 **photo5\_3** : **i0 = i1 + i2**

Nécessaire pour calculer les résistances à utiliser Oui et non / bien lire les schémas.

* résistance ( en Ohm / symbole **Ω**) = **capacité d'un matériau à s'opposer au passage du courant**

Influence intensité : diplôme « résistances » # taille rivière à un endroit.

*Loi d’ohm* : Tension = Résistance x Intensité soit **U = R x I** donc R = U/I (unités Ω = V/A)

<https://fr.wikipedia.org/wiki/Loi_d%27Ohm>

* LED (*Light Emitting Diode*) ou DEL (*Diode à Émission de Lumière*) **photo5\_4**

**diode** : un dipôle qui ne laisse passer le courant que dans **un seul sens (ou semi-conducteur)**

**🡪sens de branchement** (csq : si mal LED branchée = ne s'allumera pas + bloque le courant dans circuit)

**LED : diode qui produit de la lumière :** patte + longue = + (anode) sur 5V/ -longue =– (cathode) sur 0V

+ = borne qui distribue le courant = 5V / - (ground#terre) = borne qui absorbe le courant = 0V **photo5\_5**

Arduino : 3 pins  Gnd / 1 x 3.3 V / 1 x 5 V ( **photo5\_7-8 :** brancher LED (+) sur +3.3V pour ne pas la griller)

* breadboard (plaque d’essai) **photo5\_8**

chaque trou permet d'enficher (pour les + vieux) / plugger (pour les + jeunes) les composants

🡪 liaisons entre trous : bleus / jaunes / 5 trous de chaque colonne (rouges) liés en eux (=même point)

Rem : d’usage les 2 lignes (bleue/jaune) relier les Vin (+5 V) et les grounds (Gnd) entre eux. **photo5\_9**

* calcul de résistance (plaque d’essai) **photo5\_a**

résultat : pour LED, **moins 100 Ω (**220Ω valeur conseillée**) / +**résistance est grande, - LED éclaire.

rem : tout matériel électronique a une fiche de spécifications techniques (*datasheet*)

ex : LED produit une chute de tension entre 1 et 4 volts (selon la couleur)

hyp : tension Uled = 1,8V / alimentation Ugen = 5V / résistance = Ures\_ **photo5\_b**

calcul :

* + loi des mailles : *Ures*=*Ugen*−*Uled =* 5 V - 1,8 V = 3,2 V, tension aux bornes de la résistance.
  + iLED = 20mA (0,02A datasheet) / iArduino = 0,5A (500mA) / iPins 0 à 13 = 0,04A (40 mA) total 0.2A
  + loi d'Ohm : R=U/I donc R=3,2/0,02=160 Ω OG => choisir résistance entre 100 Ω et 1 000 Ω

ex : 220 Ω (rouge,rouge marron) / 100 Ω (marron, noir, marron) / 1 KΩ (marron, noir, rouge)

* **Montages =** un déplacement de particules chargées dans un conducteur = courant
* **2 leds / 2 résistances : photo5\_c**

rem : une avant, l’autre après sa led mais sans incidence sur l’intensité dans la led

courant : Arduino(+5V) > rouge > board > orange >résistance > led > board > orange > bleu > Gnd

* **1ère méthode : 1 LED sur un pin:** programme Blink Blink avec led sur pin 13 **photo5\_d**

|  |  |
| --- | --- |
| **sketch\_5\_led13.ino**  parcours: pin 13 > résistance > LED > Gnd  ex : modifier pour pin 7 | int pinLed=13;  void setup(){  pinMode(pinLed,OUTPUT); }  void loop(){  digitalWrite(pinLed,HIGH); // +5V  delay (1000);  digitalWrite(pinLed,LOW); // 0V  delay(1000);  } |

* **2ème méthode : 1 LED sur un pin: pour ménager Arduino photo5\_e**

méthode 1 : demande de l'énergie à l'Arduino mais préfère absorber du courant qu'en fournir

méthode 2 : monter le circuit à l'envers = Arduino(+5V) > résistance > LED > pin 13

explication : +5V dans pin 13 : risqué ?

digitalWrite(HIGH) = +5V donc différence potentiel = 0 / digitalWrite(LOW) = gnd donc courant circule

il faut inverser dans le code

**6\_ JEUX DE LUMIERES = plusieurs LEDs**

* **Projet 1 : “Blink à trois LEDs”**: faire clignoter 3LED à intervalles différents

logique : 3éteintes / 3 1s / extinction 0.1s / L3 1s / 0.1s / L2 1s / 0.1s / L1 1s / +0.8-0.6-0.4-0.2 / retour

montage : conseil : faire les branchements sans avoir connecté l'Arduino / vérifier / programme de test

pseudo-code : permet de bien voir les boucles de programmation

**montage + test : matériel : arduino + 3 leds + 3 résistances 220Ω + board + jumpers photo6\_1**

* + + simple d’abord, connecter pattes + des LEDs vers les pins (Arduino supportera la charge)

**sketch\_6-1\_3led.ino**

|  |  |
| --- | --- |
| pins 2, 4, 6  (mais éviter 0 et 1 pour affichage pour moniteur)  - delay() pour vérifier LED l’une après l’autre.  - si LED n'éclairent pas pareil :  soit résistances différentes  soit caractéristiques LED | int pinLed1, pinLed2, pinLed3;  void setup(){  //initialisation des variables  pinLed1 = 2;  pinLed2 = 4;  pinLed3 = 6;  //initialisation des modes  pinMode(pinLed1, OUTPUT);  pinMode(pinLed2, OUTPUT);  pinMode(pinLed3, OUTPUT);  //mise à 0V de chaque pin  digitalWrite(pinLed1, LOW);  digitalWrite(pinLed2, LOW);  digitalWrite(pinLed3, LOW);  }  void loop(){  //test allumage-repérage des lEDs  digitalWrite(pinLed1, HIGH);  delay(500);  digitalWrite(pinLed2, HIGH);  delay(500);  digitalWrite(pinLed3, HIGH);  delay(500);  //on éteint tout  digitalWrite(pinLed1, LOW);  digitalWrite(pinLed2, LOW);  digitalWrite(pinLed3, LOW);  delay(500);  } |

**variables & boucles ?**

Déjà 3 variables pour les pins, besoin d’une variable pour delay()

Variable globale ou locale=boucle : for (int temps=1000;temps>=200;temps=temps-200)

**sketch\_6-2\_3led\_boucle.ino**

|  |
| --- |
| int pinLed1, pinLed2, pinLed3;  void setup(){  //initialisation des variables  pinLed1 = 2;  pinLed2 = 4;  pinLed3 = 6;  //initialisation des modes  pinMode(pinLed1, OUTPUT);  pinMode(pinLed2, OUTPUT);  pinMode(pinLed3, OUTPUT);  //mise à 0V de chaque pin  digitalWrite(pinLed1, LOW);  digitalWrite(pinLed2, LOW);  digitalWrite(pinLed3, LOW);  }  void loop(){  //allumage des trois LED 1s  digitalWrite(pinLed3,HIGH);  digitalWrite(pinLed2,HIGH);  digitalWrite(pinLed1,HIGH);  delay(1000);  //on les éteint brièvement  digitalWrite(pinLed3, LOW);  digitalWrite(pinLed2, LOW);  digitalWrite(pinLed1, LOW);  delay(100);  //boucle temps qui diminue  for (int temps = 1000; temps >= 200; temps -= 200){  //les trois LEDs sont éteintes  digitalWrite(pinLed1, HIGH); //allumage LED 3  delay(temps); // pendant la valeur de temps  digitalWrite(pinLed1,LOW); //on éteint la 3  delay(100); // court délai, tout est éteint  digitalWrite(pinLed2, HIGH); //allumage LED 2  delay(temps); // pendant la valeur de temps  digitalWrite(pinLed2,LOW); //on éteint la 2  delay(100);  digitalWrite(pinLed3, HIGH); //allumage de LED 1  delay(temps); // pendant la valeur de temps  digitalWrite(pinLed3, LOW); // on éteint 1  delay(100);  //la boucle reprend  }  //retour au début de la loop();  } |

**tableaux**

rem : répétitions dans le code précédent présente des répétitions = même opération pour chaque pin.

tableau : (attention en programmation on commence à compter à partir de 0 !)

* + déclaration int pinLed[3]; (un tableau ne peut stocker que des variables du même type)
  + initialisation : int pinLed[3]={2,4,6}; ou pinLed[0]=2; pinLed[1]=4; pinLed[2]=6;

notation : [ entre ] : win = Alt Gr + ( ouAlt Gr + ) / mac = Alt + ⇧ + ( ou Alt + ⇧+ )

**sketch\_6-3\_3led\_tableau.ino**

|  |
| --- |
| int pinLed[3] = {2, 4, 6}; //déclaration et initialisation du tableau  void setup(){  //Boucle d'initialisation des modes et mise à 0V  for (int i = 0; i < 3; i++){  pinMode(pinLed[i], OUTPUT); //on utilise les valeurs du tableau  digitalWrite(pinLed[i], LOW); // l'une après l'autre  }  }  void loop(){  //on allume les 3 LED (ici en utilisant une boucle)  for (int i = 0; i < 3; i++) {  digitalWrite(pinLed[i], HIGH);// on allume chaque led  }  delay(1000); //pendant 1 seconde  //puis on les éteint brièvement  digitalWrite(pinLed[0],LOW); // on éteint la 1  digitalWrite(pinLed[1], LOW); // on éteint la 2  digitalWrite(pinLed[2],LOW); // on éteint la 3  delay(100);//délai bref    //Boucle pour diminuer le temps d’allumage des LED  for (int temps = 1000; temps >= 200; temps -= 200) {  //Les trois LED sont éteintes  for (int i = 0; i < 3; i++) {  // t pour parcourir le tableau  digitalWrite(pinLed[i], HIGH);//on allume la LED correspondante  delay(temps); // pendant la valeur de temps  digitalWrite(pinLed[i], LOW); //on éteint la correspondante  delay(100); // court délai pendant lequel les 3 LED sont éteintes  }  // 3 LED allumées pendant une durée “temps”, boucle reprend décrémentée  }  } |

* **Projet 2 : une GuirLED**: à partir de 5 LEDs de couleurs choisies (positionnement rapproché)

logique : "figures" lumineuses variées

* Les LED s'allument de droite à gauche une par une ;
* Les LED s'allument de gauche à droite une par une ;
* Les LED s'allument toutes puis s'éteignent de droite à gauche ;
* Les LEDs s'allument toutes puis s'éteignent de gauche à droite.

tableaux : valeur des pins / booléens pour les états d’allumage

tableau d’état des leds :1 signifie allumé, le 0 éteint :

| Séquence | Led1 | Led2 | Led3 | Led4 | Led5 |
| --- | --- | --- | --- | --- | --- |
| 0 | **0** | **0** | **0** | **0** | **0** |
| 1 | **0** | **0** | **0** | **0** | **1** |
| 2 | **0** | **0** | **0** | **1** | **0** |
| 3 | **0** | **0** | **1** | **0** | **0** |
| 4 | **0** | **1** | **0** | **0** | **0** |
| 5 | **1** | **0** | **0** | **0** | **0** |

principe tableau de tableaux : int tableau[5][5]= {2,4,5,7,9,3,4,6,8,0,3,4,1,2,5,4,5,6,8,3,2,2,2,4,5};

ex : tableau [3][4] = position 4 du tableau 3 qui vaut 3 ici -> nécessité de 2 boucles

remarques : nécessiter à initialiser avec déclaration / revenir à la ligne pour plus facile à lire

🡪ici = tableau de 6 séquences, qui contient tableau de 5 booléens pour chaque séquence

montage **photo6\_2** 5 LEDs connectées sur les pins 2,4,6,8 et 10

programme remarque : ce serait encore mieux avec des fonctions **sketch\_6-4\_guirlande\_led.ino**

|  |
| --- |
| int pinLed[5]={2,4,6,8,10}; // Tableau listant les pins  //tableau états (booléen 1 allumé, 0 éteint) des LED à chaque séquence  boolean affichage[25][5]={  0,0,0,0,0,  0,0,0,0,1,  0,0,0,1,0,  0,0,1,0,0,  0,1,0,0,0,  1,0,0,0,0,  0,0,0,0,0,  1,0,0,0,0,  0,1,0,0,0,  0,0,1,0,0,  0,0,0,1,0,  0,0,0,0,1,  0,0,0,0,0,  1,1,1,1,1,  1,1,1,1,0,  1,1,1,0,0,  1,1,0,0,0,  1,0,0,0,0,  0,0,0,0,0,  1,1,1,1,1,  0,1,1,1,1,  0,0,1,1,1,  0,0,0,1,1,  0,0,0,0,1,  0,0,0,0,0};  void setup() {  for (int i=0;i<5;i++)  {  pinMode(pinLed[i],OUTPUT);  digitalWrite(pinLed[i],LOW);  }    }  void loop() {  for (int i=0;i<25;i++) // boucle de séquence d'affichage  {  for (int p=0;p<5;p++) // boucle pour chaque pin  {  boolean etat=affichage[i][p]; // on va chercher l'état pour le pin  digitalWrite(pinLed[p],etat); // on met le pin concerné à l'état  }  //tous les pins sont dans l'état de la séquence en cours  delay(300); //petite pause d'affichage  // on passe à la séquence suivante  }  // fin des séquences et on repart au début de la loop()  } |

*Exercice : passer à 10 LED et faire des séquences plus complexes.*

Remarque : possible aussi de positionner les diodes différemment (en carré, cercle, ou autre).

**7\_ BOUTON POUSSOIR** : ajout d’interactivité en entrée = « le sens du toucher »

* **les entrées numériques** reçoivent  du +5V ou du 0V

sorties : digitalWrite(pin, état) / entrées : digitalRead(pin)

affectation pin : un pin est soit en entrée, soit en sortie, mais pas les deux 🡪 pinMode(pin, mode);

* pinMode(pin,OUTPUT); = **mode écriture =** envoye ou non du courant = sortie
* pinMode(pin,INPUT); = **mode lecture =** à l'écoute du courant qui arrive = entrée

exemple : code permet de lire la valeur reçue par le pin 10

|  |
| --- |
| void setup() {  Serial.begin(9600); //Initialisation communication avec le moniteur série  pinMode(10,INPUT); //On indique à l’Arduino le mode du pin (entrée)  }  void loop() {  boolean a=digitalRead(10);// et on l’affecte à la variable "a"  Serial.println(a); // on l'affiche sur le moniteur  } |

* **bouton poussoir** lorsque l'on appuie, le courant passe & inversement (mais il faut garder la pression)

rem : limite analogie rivière : si le circuit n’est pas fermé, le courant ne s’accumule pas = pas de courant

**photo7\_1 bouton** dipôle (4 pattes reliées 2 à 2 \_ tester à chaque fois pour savoir) / **photo7\_2 circuit**

* **montage sans programmation photo7\_3**

connection directe Arduino-bouton poussoir-LED (tourner ¼ de tour si toujours allumé ,-)

* **montage avec programmation photo7\_4** en le reliant à un pin qui est en mode lecture

|  |
| --- |
| int pinBouton;  void setup(){  Serial.begin(9600);  pinBouton=10;  pinMode(pinBouton,INPUT);  }  void loop(){  boolean etatBouton=digitalRead(pinBouton);  Serial.println(etatBouton);  } |

résultat : afficher0, puis 1 lorsqu'on appuie sur le bouton mais reste à 1 en relâchant.

explication : côté erratique d’Arduino car pin 10 n’est connecté à rien, quand le bouton est levé

solution : forcer l'Arduino à lire quelque chose = ajouter d'une résistance...

* **montage avec** **résistance pull-down** **: photo7\_5 circuit / photo7\_6** (autre façon de monter bouton)

électricité : passage du courant = choisit le chemin qui lui résiste le moins

principe : on ajoute une résistance assez forte pour que le courant ne passe que s'il y est obligé (**10kΩ**).

explication :

* + poussoir=baissé, courant du +5V au pin10 = indique HIGH (1) = ne vas pas vers ground car effort 10kΩ
  + poussoir=levé, très faible courant résiduel du pin10 absorbé par Gnd = indique LOW (0)

remarque : montage à connaître car prévoir comportement erratique quel que soit le type de contacteur

résistances : voir doc pour codification (trouver un lien)

* **montage avec** **résistance pull-up** **: photo7\_7 circuit / photo7\_8** (autre façon de monter bouton)

électricité : passage du courant = choisit le chemin qui lui résiste le moins

principe : monter la résistance, non pas vers le ground, mais vers le +5V donc connecter poussoir au ground

explication : opposé du fonctionnement en pull-down (= mode ECRITURE car fermé=HIGH)

* + poussoir=fermé, +5V & pin10 absorbés par le ground = indique LOW = mode LECTURE
  + poussoir=levé, +5V nourrit le pin10 = indique HIGH
* **montage avec** **mode INPUT\_PULLUP** **: photo7\_9** (autre façon de monter bouton)

arduino : propose par défaut d'activer une résistance de 20 KΩ interne = résistance pull-up

🡪 **avantage = pas besoin d’une résistance en plus**

code : indiquer Arduino d'activer cette résistance = pinMode(pin,INPUT\_PULLUP);

🡪 commande pinMod préparer un pin dans 3 modes différents :

* + Mode **OUTPUT** = Arduino fournit par programmation du +5V ou du 0V.
  + Mode **INPUT** = Arduino lit état du pin HIGH (1) pour +5 V reçus / LOW (0) pour 0 V
  + Mode **INPUT\_PULLUP** = Arduino lit infos reçues, mais pin connecté en interne résistance de 20KΩ

remarque : **pin 13 en lecture : mode INPUT\_PULLUP déconseillé**, préféer resistance externe pull-up/down

pin 13 lié à 1LED+1 résistance (série) qui baissent tension => fournit pas +5V, mais +1.7V = toujours LOW

* **Programme "Jour/Nuit" photo7\_a**

montage : 1 poussoir connecté au pin 2 avec résistance 10kΩ en pull-down / LED1 pin 4 & LED2 pin 6 avec 220Ω

principe : tester l'état du bouton poussoir (pin en lecture) et faire agir le programme en fonction :

* + bouton=levé, LED1 allumée, LED2 éteinte
  + bouton=appuyé, LED1 éteinte, LED2 allumée

|  |
| --- |
| int pinBouton, pinLed1, pinLed2;  void setup(){  //initialisation des variables  pinBouton = 2;  pinLed1 = 4;  pinLed2 = 6;  //définition des modes  pinMode(pinBouton, INPUT);  pinMode(pinLed1, OUTPUT);  pinMode(pinLed2, OUTPUT);  }  void loop(){  //lecture de l'état du bouton et stockage dans etatBouton  boolean etatBouton = digitalRead(pinBouton);  //test des conditions  if (etatBouton){ //si bouton appuyé (donc 1)  digitalWrite(pinLed1,LOW);  digitalWrite(pinLed2,HIGH);  }  else { //sinon  digitalWrite(pinLed1,HIGH);  digitalWrite(pinLed2,LOW);  }  delay(100);  } |

code :

* écritures identiques : if (etatBouton=HIGH) # if (etatBouton==1) # if (etatBouton)
* delay(100); : conseillé toujours laisser temps à Arduino avant de refaire boucle, surtout si lecture données
* tests combinés pour éviter succession 2 conditions :

if (test) { //code à exécuter } else if (autreTest) { //autre code} else { //code exécuté si aucun test vérifié }

* **Programme "Interrupteur" photo7\_b**

montage : 1 poussoir connecté au pin 2 en mode INPUT\_PULLUP / LED pin résistance 220Ω

principe : créer une variable qui change à chaque appui sur le bouton :

* + bouton=appuyé (1fois), LED s’allume et reste allumée
  + bouton=appuyé à nouveau, LED s’éteint et reste éteinte
  + bouton=levé, LED1 allumée, LED2 éteinte
  + LED2 allumée

|  |
| --- |
| int pinBouton, pinLed;  boolean etatAllumage;  void setup(){  //initialisation des variables  Serial.begin(9600);  pinBouton = 2;  pinLed = 4;  etatAllumage=0;  //définition des modes  pinMode(pinBouton, INPUT\_PULLUP);  pinMode(pinLed, OUTPUT);  }  void loop(){  Serial.print(etatAllumage);  if (etatAllumage) { //on teste si etatAllumage est à 1  digitalWrite(pinLed, HIGH);//on allume la LED  }  else { //sinon  digitalWrite(pinLed, LOW); //on éteint la LED  }  //lecture de l'état du bouton et stockage dans etatBouton  boolean etatPinBouton = digitalRead(pinBouton);  Serial.println(etatPinBouton);  //test des conditions  if (!etatPinBouton) { //si bouton appuyé (pin=0 car mode INPUT\_PULLUP)  if (etatAllumage) { //si etatAllumage à 1  etatAllumage=0; //on le passe à 0  }  else { //sinon  etatAllumage=1; //on le passe à 1  }  }  delay(200);  } |

code : écritures identiques : !etatBouton # etatPinBouton!=1 # etatBouton==0

remarque : pb rapidité machine vs humain : si maintenu appuyé, boucle se répète / passe modif variable

🡪 ajouter 1 variable pour tester si le bouton a été relâché entre-temps

**8\_ FONCTIONS & RANDOM**

* **Programme "dé électronique"** de 0 à 5 (plutôt que de 1 à 6)
* **Étape 1 : affichage d’un nombre aléatoire sur la console** **:** random(max); / random(min,max);

|  |
| --- |
| int nbAlea;  void setup() {  Serial.begin(9600);  }  void loop() {  nbAlea=random(100); //construit le nombre aléatoire  Serial.println(nbAlea);//affichage du nombre  delay(500);//attente  } |

Résultat : nombres aléatoires entre 0 et 99 (puisque 100 n'est pas compris dedans).

Problème : si 'reset' Arduino : même série de nombres = pas vraiment aléatoire ! = **pseudo-aléatoires**

Solution : randomSeed(unNombre); , avec unNombre, int hasard !!! (seed en anglais)

entrées analogiques Arduino = seule méthode choix tranquille de la variable  unNombre .

🡪 assigne une entrée analogique à "unNombre  par la méthode randomSeed(analogRead(0));

Attention : le pin A0 ne doit pas être connecté

|  |
| --- |
| int nbAlea;  void setup() {  Serial.begin(9600);  randomSeed(analogRead(0));//initialise la séquence aléatoire  }  void loop() {  nbAlea=random(100); //construit le nombre aléatoire  Serial.println(nbAlea);//affichage du nombre  delay(500);//attente  } |

Résultat : si on appuie sur ‘reset’, séquences différentes

* **Étape 2 : montage : photo8\_1 / 8\_2 montage**

Principe : 5 leds en série avec résistance 220 Ω sur pins 2, 3, 4, 5, 6 (rappel : patte courte sur gnd)

Commentaires :

* + fils oranges connexions Arduino / rouge +5V (non utilisé) / bleu ground
  + patte + diodes reliée à un pin Arduino / chaque patte - reliée au ground en passant par résistance 220Ω
  + simplification : connecter directement résistance ground (rail du haut ou du bas en bleu)
* **Étape 3 : test des LEDs :**

Astuce  : on teste le nombre 5 car permet allumer toutes les LEDs !

Principe : chaque pin : 1 variable / en mode OUTPUT (envoi de courant) / positionnement HIGH (1)

|  |  |
| --- | --- |
| Version 0 | Version 1 (simplifiée) |
| int ledHautGauche=2;  int ledBasGauche=3;  int ledCentre=4;  int ledHautDroite=6;  int ledBasDroite=5;  void setup() {  //positionnement des pin en OUTPUT  pinMode(ledHautGauche,OUTPUT);  pinMode(ledBasGauche,OUTPUT);  pinMode(ledCentre,OUTPUT);  pinMode(ledHautDroite,OUTPUT);  pinMode(ledBasDroite,OUTPUT);    //allumage de tous les pins  digitalWrite(ledHautGauche,HIGH);  digitalWrite(ledBasGauche,HIGH);  digitalWrite(ledCentre,HIGH);  digitalWrite(ledHautDroite,HIGH);  digitalWrite(ledBasDroite,HIGH);  }  void loop() {  //inutile pour ce test  } | void setup() {  for (int l=2;l<7;l++){// boucle 2 à 6  pinMode(l,OUTPUT);// mode OUTPUT  digitalWrite(l,HIGH);// allumage  }  }  void loop() {  //inutile pour ce test  } |

Résultat  : 5 LED allumées sinon vérifier programme, connexions, sens des LED, résistances

Remarque  : pour vérifier le 0, tout mettre en position LOW

* **Étape 4 : dé :**

Codification : nombres & pins (HIGH ou LOW) :

| Nombre | ledHautGauche | ledBasGauche | ledCentre | ledHautDroite | ledBasDroite |
| --- | --- | --- | --- | --- | --- |
| 0 | LOW | LOW | LOW | LOW | LOW |
| 1 | LOW | LOW | HIGH | LOW | LOW |
| 2 | HIGH | LOW | LOW | LOW | HIGH |
| 3 | LOW | HIGH | HIGH | HIGH | LOW |
| 4 | HIGH | HIGH | LOW | HIGH | HIGH |
| 5 | HIGH | HIGH | HIGH | HIGH | HIGH |

Code :

|  |  |
| --- | --- |
| Algorithme //à répéter = loop | Programme |
| for (int l=0;l<6;l++){  if (l==0){  //affichage en forme de 0  }  if (l==1){  //affichage en forme de 1  }  if (l==2){  //affichage en forme de 2  }  if (l==3){  //affichage en forme de 3  }  if (l==4){  //affichage en forme de 4  }  if (l==5){  //affichage en forme de 5  }  } | //définition des variables de pin pour chaque LED  int ledHautGauche = 2;  int ledBasGauche = 3;  int ledCentre = 4;  int ledHautDroite = 6;  int ledBasDroite = 5;  void setup() {  //initialisation des pins  for (int l = 2; l < 7; l++) { //boucle 2à6  pinMode(l, OUTPUT); // mode OUTPUT  digitalWrite(l, LOW); // tout est éteint  }  }  void loop() {  for (int l = 0; l < 6; l++) {  if (l == 0) {  //affichage des diodes en forme de 0  digitalWrite(ledHautGauche, LOW);  digitalWrite(ledBasGauche, LOW);  digitalWrite(ledCentre, LOW);  digitalWrite(ledHautDroite, LOW);  digitalWrite(ledBasDroite, LOW);  }  if (l == 1) {  //affichage des diodes en forme de 1  digitalWrite(ledHautGauche, LOW);  digitalWrite(ledBasGauche, LOW);  digitalWrite(ledCentre, HIGH);  digitalWrite(ledHautDroite, LOW);  digitalWrite(ledBasDroite, LOW);  }  if (l == 2) {  //affichage des diodes en forme de 2  digitalWrite(ledHautGauche, HIGH);  digitalWrite(ledBasGauche, LOW);  digitalWrite(ledCentre, LOW);  digitalWrite(ledHautDroite, LOW);  digitalWrite(ledBasDroite, HIGH);  }  if (l == 3) {  //affichage des diodes en forme de 3  digitalWrite(ledHautGauche, LOW);  digitalWrite(ledBasGauche, HIGH);  digitalWrite(ledCentre, HIGH);  digitalWrite(ledHautDroite, HIGH);  digitalWrite(ledBasDroite, LOW);  }  if (l == 4) {  //affichage des diodes en forme de 4  digitalWrite(ledHautGauche, HIGH);  digitalWrite(ledBasGauche, HIGH);  digitalWrite(ledCentre, LOW);  digitalWrite(ledHautDroite, HIGH);  digitalWrite(ledBasDroite, HIGH);  }  if (l == 5) {  //affichage des diodes en forme de 5  digitalWrite(ledHautGauche, HIGH);  digitalWrite(ledBasGauche, HIGH);  digitalWrite(ledCentre, HIGH);  digitalWrite(ledHautDroite, HIGH);  digitalWrite(ledBasDroite, HIGH);  }  delay(1000); //attente d'une seconde  }  // à répéter à l'infini  } |

Remarque  : code répétitif … d’où intérêt des fonctions

* **fonctions**
* Principe  : **photo8\_3**
  + appel fonction1 depuis pgm principal = interruption pgm principal pour exécuter la fonction puis retour.
  + 2 écritures : 1/**définition (déclaration) de la fonction**=code à exécuter 2/ **appel à la fonction**
* Déclaration  : même forme que le  setup ou la  loop (qui sont en fait des fonctions !)

type nomDeLaFonction(){ //code du programme de la fonction }

* + type = type de la variable que la fonction peut renvoyer (“void”=“vide”, ne renvoie rien)
  + nomDeLaFonction = nom choisi de la fonction
  + ( ) = obligatoires, servent à envoyer des paramètres à la fonction s’il y en a.
  + { } = code du programme de la fonction (pas de point-virgule après l'accolade de fermeture)

Remarque : placer les fonctions après la fonction loop()

* Appel  : nomDeLaFonction(); dans le programme principal quand on en a besoin
* Démo  : tester le programme en activant la console

|  |
| --- |
| void setup() {  Serial.begin(9600);  }  //boucle de programme principal  void loop() {  Serial.println("Je suis dans le programme principal, en début de boucle");  delay(1000);  fonction1();  fonction2();  }  //code de la fonction1  void fonction1(){  Serial.println("Je suis dans la fonction1");  delay(1000);  }  //code de la fonction2  void fonction2(){  Serial.println("Je suis dans la fonction2");  delay(1000);  } |

* Passage de paramètre(s) à la fonction  : une ou plusieurs variables ou constantes
  + Déclaration : type nomDeLaFonction(type unParametre) { //code de la fonction }
  + Appel : nomDeLaFonction(parametre); paramètre envoyé et déclaré doivent être de même type
  + Déclaration : type nomDeLaFonction(type param1,type param2,...){ code de la fonction }
  + Appel : nomDeLaFonction(param,autreParam, ,...);

Remarque : une fonction ne peut retourner qu'une seule valeur (de type défini avant le nom de la fonction).

|  |
| --- |
| void setup(){  Serial.begin(9600);  }  void loop(){  //appel de fonction avec variables pré-déclarées  int a=5;  int b=3;  int s=somme(a,b);  Serial.println(s);  //appel de fonction en paramètres directs  Serial.println(produit(5,2));  }  //déclaration des fonctions  int somme(int nb1,int nb2){  //utilisation de variables dans la fonction  int res=nb1+nb2;  return res;  }  int produit(int nb1,int nb2){  //retour direct d'un résultat sans variable intermédiaire  return nb1\*nb2;  } |

* **Fonctions pour le dé à 5 leds**
* Algorithme :

|  |
| --- |
| //définition des variables  void setup(){  //on place les pins en mode OUTPUT  }  void loop(){  for (t=0;t<6;t++){  //on teste le nombre t et on envoie le programme à la bonne fonction  }  }  //définition des fonctions  void afficheZero(){  //code pour tout éteindre  }  void afficheUn(){  //code pour le 1  }  void afficheDeux(){  //code pour le 2  }  void afficheTrois(){  //code pour le 3  }  void afficheQuatre(){  //code pour le 4  }  void afficheCinq(){  //code pour le 5  } |

* Allumage des diodes pour chaque face de dé  :

|  |
| --- |
| //déclaration des variables  int ledHautGauche=2;  int ledBasGauche=3;  int ledCentre=4;  int ledHautDroite=6;  int ledBasDroite=5;  //fonction d'initialisation  void setup() {  //mise en mode OUTPUT des pins 2 à 6 et positionnement en LOW  for (int t=2;t<7;t++){  pinMode(t,OUTPUT);  digitalWrite(t,LOW);  }  }  //boucle principale  void loop() {  // boucle pour faire varier le nombre  for (int nb=0;nb<6;nb++){  affichage(nb);//appel de la fonction d'allumage des LEDs  delay(500);  }  }  // déclaration des fonctions  //cette fonction sert à éteindre toutes les diodes  void setZero(){  for (int t=2;t<7;t++){  digitalWrite(t,LOW);  }  }  //cette fonction récupère un nombre et allume les LED en conséquence  void affichage(int nombre){  setZero();//appel de la fonction qui éteint toutes les LED    //il suffit maintenant d'allumer les bonnes diodes  //en testant la valeur de 'nombre'  if (nombre==1){  digitalWrite(ledCentre,HIGH); //on allume la diode du centre  return;//sortie de la fonction  }  if (nombre==2){  //on allume les diodes haut/droite et bas/gauche  digitalWrite(ledHautDroite,HIGH);  digitalWrite(ledBasGauche,HIGH);  return;//sortie de la fonction  }  if (nombre==3){  //on allume les diodes centre, haut/gauche, bas/droite  digitalWrite(ledHautDroite,HIGH);  digitalWrite(ledCentre,HIGH);  digitalWrite(ledBasGauche,HIGH);  return;//sortie de la fonction  }  if (nombre==4){  //on allume toutes les diodes sauf celle du centre  digitalWrite(ledHautGauche,HIGH);  digitalWrite(ledHautDroite,HIGH);  digitalWrite(ledBasGauche,HIGH);  digitalWrite(ledBasDroite,HIGH);  return;//sortie de la fonction  }  if (nombre==5){  //on allume toutes les diodes  digitalWrite(ledHautGauche,HIGH);  digitalWrite(ledBasGauche,HIGH);  digitalWrite(ledHautDroite,HIGH);  digitalWrite(ledBasDroite,HIGH);  digitalWrite(ledCentre,HIGH);  return;//sortie de la fonction  }  //inutile de tester le 0 car on a commencé par tout éteindre  return;//sortie de la fonction  } |

Remarques :

* + return permet de renvoyer une valeur au programme principal
  + on peut appeler une fonction d’une autre fonction
  + scope des variables : variable **créée dans** **fonction** pas accessible en dehors / variable globale accessible
* Programme final pour Dé à 5 LEDs  : pas la solution, juste guide programmation

|  |
| --- |
| //déclaration des variables  //LEDs sur les pins 2 à 6  //bouton sur le pin 7    //initialisation  void setup(){  //appel de la fonction randomSeed() pour la séquence aléatoire  //pin 2 à 6 en mode OUTPUT  //pin 7 en mode INPUT\_PULLUP  //appel de la fonction setZero() pour tout éteindre  }  //boucle principale  void loop(){  //test de l'état du bouton avec boolean etatBouton  if (le bouton est cliqué){  for... //boucle for qui temporise l'affichage de plus en plus lentement  {  //exctinction brève des LEDs (100ms)  //tirage d'un nombre aléatoire avec la fonction random()  //appel de la fonction d'allumage des LEDs  //temporisation  }//sortie de la boucle de temporisation  }//sortie du test bouton  }  //déclaration de la fonction setZero()  void setZero(){  //code pour tout éteindre  }  //déclaration de la fonction d'allugmage  void allumage(int nombre){  //code pour l'affichage en fonction du nombre  } |

**9\_ POTENTIOMETRE / ENTREES ANALOGIQUES / MAPPAGE**

* **Potentiomètre analogique :** une forme de résistance qui varie (2 types : **linéaires** et **logarithmiques) photo9\_1**

entrées numériques = ne prennent que deux valeurs : 1 ou 0 <> entrées analogiques = données électriques plus subtiles.

Principe : composant à 3 pattes = un curseur se déplace sur une piste résistante et permet de faire varier la résistance

= l'électricité entre par un côté et sort par le curseur (curseur proche de l'entrée = résistance est faible) **photo9\_2**

**🡪 à intensité égale, en faisant varier la résistance, on fait varier la tension aux bornes**

Branchement : connecter sur la patte du curseur + l'une patte sur le +5V et l'autre sur le ground.

Rem : la 3e patte donne résistance « complémentaire » = si branchée vers alim et 2 autres vers lampes = 1+2- à 1-2+

Exemple (potentiomètre linéaire) : résistance maximum 10 KΩ : entrèe = 0Ω / milieu 5 KΩ / 10 kΩ (valeur max).

Modèles : mouvement droits (déplace le curseur sur une ligne) / circulaires (en le faisant tourner)

* Montage test = **2 diodes en opposition**  : **photo9\_3**

Principe : en faisant varier le potentiomètre, la luminosité d'une diode pendant que celle de l'autre diminue

* + connecter résistance vers +5 V, patte centrale du potentiomètre à la résistance.
  + connecter ensuite chaque autre patte du potentiomètre à la borne + d’une diode, et chaque diode au ground.
  + On garde la résistance de 220 Ω pour protéger les diodes quand le potentiomètre est au minimum.
* **Entrées analogiques**  : **photo9\_3**

**signal analogique = peut prendre infinité de valeurs entre 2 bornes <> signal numérique nombre précis de valeurs**

ex : **photo9\_4 circuit / photo9\_5 résultat**

* + **analogique** : si tension de 0 V à 5 V = nombre infini valeurs entre 0 et 5V (attention : 0Ω fait un court circuit !!!)
  + **numérique =** nombre de valeurs définies (principe de l'escalier = progression par cran = **résolution** du signal

*vocabulaire*:

* + *quantification* = chercher la valeur numérique d'un signal analogique
  + *CAN (convertisseur analogique numérique)* transforme un signal analogique en signal numérique

= Arduino en à 6 : transformer jusqu'à 6 signaux analogiques en signaux numériques = nombres entiers

* + *résolution* du signal numérique : taille des crans de l’escalier

= sa limite de définition est de 1024 valeur possibles entre 0 et 1023 (et signal reçu ne peut et supérieur à 5V)

**photo9\_6 Arduino \_ entrées analogiques :** A0, A1, A2, A3, A4, et A5

* **montage conversion analogique-numerique**

Principe : curseur sur A0, 2 autres sur 5V et ground **photo9\_7 circuit**

* + courant part du +5V, entre dans le potentiomètre, sort par le curseur vers le pin A0 de l'Arduino.
  + connexion au ground permet la lecture de la valeur de position du curseur.

Rem : Si oubliez connexion au +5V = uniquement la valeur 0 / si pas connecté au ground = 1023 quelque soit position

Montage : **photo9\_8 montage** potentiomètre rotatif de valeur 10 kΩ

* Programme : **afficher la valeur d'un potentiomètre connecté au pin A0**

langage : analogRead(pin); pin vaut : A0, A1, A2, A3, A4, A5 ou aussi 0, 1, 2, 3, 4,5 et int entre 0 et 1023

analogRead(2) <> digitalRead(2) mais même utilisation possible (ex : pinMode(A0,OUTPUT); )

|  |
| --- |
| int pinPot=0; //variable pour définir le CAN où est connecté le potentiomètre  int valPot=0; //variable pour récupérer la tension aux bornes du potentiomètre traduite par le CAN . On l’initialise à 0.  void setup() {  Serial.begin(9600); //Initialisation de la communication avec la console  }  void loop() {  valPot=analogRead(A0); //lit la tension, la convertit en valeur numérique et la stocke dans valeurPot  Serial.print("Valeur lue : ");  Serial.println(valPot);  } |

Remarque :valPot initialisée à 0 = bonne habitude d'initialiser ses variables pour éviter valeurs incohérentes

Résultats : précision (résolution) largement suffisante, très difficile de réussir à faire progresser de 1 en 1.

* **clignotant à vitesse variable**

Principe : LED clignote +/- vite selon curseur : 5 valeurs de 1Hz (0,5s=1/0,5s=0) au +lent / 5Hz (0,1s-0,1s) au +rapide

Montage : diode pin 8 (avec résistance) et potentiomètre pin A0.

Programme : difficulté réside dans l'obligation des temps de clignotement (solutions : calcul ou tests if)

* + Étape 1 : **clignotement sans l'obligation des temps de clignotement**

= récupère la valeur du potentiomètre -> temps d'attente allumage/extinction.

|  |
| --- |
| int pinPot=0; //variable pour le CAN utilisé  int valPot=0; //variable qui va stocker la tension lue. On l'initialise à 0.  int pinLED=8; // pin de connexion de la LED  void setup() {  pinMode(pinLED,OUTPUT); //Mode OUTPUT pour le pin de LED  digitalWrite(pinLED,HIGH);//On allume la LED  }  void loop() {  valPot=analogRead(A0); //lit tension, la transforme, la stocke  int attente=valPot; //le délai d'attente est la valeur du potentiomètre  digitalWrite(pinLED,HIGH); //on allume la LED  delay(attente); //on attend en fonction de la variable attente  digitalWrite(pinLED,LOW); //on éteint la LED  delay(attente); //on attend  } |

* + Étape 2 : **variation de l'attente soit par paliers**

Principe :tableau de valeur d'attente : valeurs entre 0 et 1023 & 5 clignotement = 1024/5=204.8#205

| Valeur du potentiomètre | Valeur d'attente |
| --- | --- |
| entre 0 et 204 | 1000/5/2=100ms |
| entre 205 et 408 | 1000/4/2=125ms |
| entre 409 et 613 | 1000/3/2=167ms (on arrondit) |
| entre 614 et 818 | 1000/2/2=250ms |
| entre 818 et 1023 | 1000/1/2=500ms |

Remarques :valeur d’attente calculée selon la même règle : = int(1000/valInter/2), nb clignotement/s

|  |  |
| --- | --- |
| Proposition 1 | Proposition 2 |
| int pinPot=0; //variable pour le pin où est connecté le potentiomètre  int valPot=0; //variable pour récupérer la valeur lue. On l'initialise à 0.  int pinLED=8;  void setup() {  pinMode(pinLED,OUTPUT);  digitalWrite(pinLED,HIGH);  }  void loop() {  valPot=analogRead(A0); //lit la valeur, la transforme et la stocke dans la variable  int attente=500;  if (valPot>=0 && valPot<=204){  attente=100;  }  if (valPot>=205 && valPot<=408){  attente=125;  }  if (valPot>=409 && valPot<=613){  attente=167;  }  if (valPot>=614 && valPot<=818){  attente=250;  }  if (valPot>=818 && valPot<=1023){  attente=500;  }  digitalWrite(pinLED,HIGH);  delay (attente);  digitalWrite(pinLED,LOW);  delay(attente);  } | int pinPot=0; //variable pour le pin où est connecté le potentiomètre  int valPot=0; //variable pour récupérer la valeur lue. On l'initialise à 0.  int pinLED=8;  void setup() {  pinMode(pinLED,OUTPUT);  digitalWrite(pinLED,HIGH);  }  void loop() {  valPot=analogRead(A0); //lit la valeur, la transforme et la stocke dans la variable  int attente=100;  if (valPot>817){  attente=500;  }  else if (valPot>613){  attente=250;  }  else if (valPot>408){  attente=167;  }  else if (valPot>204){  attente=125;  }  digitalWrite(pinLED,HIGH);  delay (attente);  digitalWrite(pinLED,LOW);  delay(attente);  } |

Remarques sur proposition 2

* + un else if qui permet de passer aux autres tests si le test précédent n'est pas vérifié

= plus rapide de comparer a>b que a>=b

* + si aucun test vérifié, variable = valeur assignée au début de loop (valeur 500 ici)
  + Étape 3 : **code final**

Calcul : la valeur d’attente peut se définir comme une variable : attente = int(1000/valInter/2)

🡪 donc calculer  int valInter=int(valPot\*5/1024)+1

Explications : préfixe int convertit en int / 1024 res entre 0 & 4 (1023 entre 0 et 5) / +1 entier supérieur

🡪 donc int attente=int(1000/valInter/2) soit 1 seconde divisée nb clignotements (/2 allumé-éteint)

|  |
| --- |
| int pinPot=0; //variable pour le pin où est connecté le potentiomètre  int valPot=0; //variable pour récupérer la valeur lue. On l'initialise à 0.  int pinLED=8;  void setup() {  pinMode(pinLED,OUTPUT);  digitalWrite(pinLED,HIGH);  }  void loop() {  valPot=analogRead(A0); //lit la valeur, la transforme et la stocke dans la variable  int valInter=int(valPot\*5/1024)+1; //produit en croix  int attente=1000/valInter/2; // calcul du temps d'arrêt  digitalWrite(pinLED,HIGH);  delay (attente);  digitalWrite(pinLED,LOW);  delay(attente);  } |

* **mappage de valeurs** : map(valeur,min,max,transMin,transMax); renvoie toujours un int

légende : valeur , min-max = plage de valeurs possible, transMin et  transMax = plage de valeurs cible

exemple : int valInter=map(valPot,0,1023,1,5);

|  |
| --- |
| int pinPot=0; //stocke le CAN utilisé  int valPot=0; //variable pour récupérer la valeur de tension du potentiomètre. On l'initialise à 0.  int pinLED=8;  void setup() {  pinMode(pinLED,OUTPUT); //mode OUTPUT pour le pin de LED  digitalWrite(pinLED,HIGH);//on allume la LED  }  void loop() {  valPot=analogRead(A0); //lit la valeur de la tension, la numérise et la stocke dans valPot  int valInter=map(valPot,0,1023,1,5); //fonction de mappage  int attente=1000/valInter/2; // calcul du temps d'arrêt  digitalWrite(pinLED,HIGH); //on allume la LED  delay (attente); //attente calculée  digitalWrite(pinLED,LOW); //on éteint la LED  delay(attente); //même attente  } |

Remarque : peut servir à inverser les valeurs

|  |  |  |
| --- | --- | --- |
| Appel de la fonction map() | Action | Résultat |
| int valRes=map(128,0,1000,10,0); | Transforme la valeur 128, initialement située entre 0 et 1000, en une valeur entre 10 et 0 | valRes=9 |
| int valRes=map(128,0,1000,-10,10); | Transforme la valeur 128, initialement située entre 0 et 1000, en une valeur entre -10 et 10 | valRes=-8 |

Attention : calculs effectués en partie entière à chaque opération => différences possibles avec vrai résultat

* **joystick** : **photo9\_9 montage** 2 potentiomètres (+ bouton poussoir inclus pression verticale)

**avant**: joystick seulement 4 positions (4 interrupteurs simples) = envoi 1 résultat

**auj**: potentiomètre permet échantillonnage numérique = gagne en précision

Principe : récupérer 2 valeurs sur entrées analogiques puis mapper selon le besoin

**10\_ SERVO-MOTEUR**

* **Servo-moteur photoa\_1** servo : du latin "servus", esclave, moteur à courant continu

Matériel : un bloc, un axe et une roue trouée = tourne quand est parcouru par de l’électricité

Moteur : moteur associé à une série d'engrenages => gain de puissance mais vitesse rotation réduite

Engrenages : 2 principes : **photoa\_2** (petite roue = ‘*pignon*’, rapport grande/petite = ‘*couple*’)

* si roue 1 tourne dans un sens, roue 2 tourne dans l’autre sens
* la taille respective des roues implique la vitesse (ex : R1 6 dents = 4 tours, R2 24 dents = 1 tour)

🡪 moteur du servo entraîne engrenages = gagner en puissance rotation = tourne moins vite mais gagne couple

Couple : effort de rotation applicable sur un axe = force est nécessaire le faire tourner **photoa\_3**

Légende : axe=celui servo-moteur + bras partie solide pour fixer qqch : poids=force appliquée sur le bras

🡪 distance axe-position poids= +poids éloigné axe, +effort nécessaire important (capacité d'effort = couple)

OG : 1,5 kg.cm = servo peut soulever poids 1,5 kg placé à 1cm de son axe (si déplace 10cm, couplex10 =15kg.cm)

Attention : si le couple nécessaire à la rotation est supérieur au couple qu'il fournit = risquer endommager

**Définition : servo-moteur a un axe qui fournit rotation d’une force (couple) (à ne pas dépasser)**

Servo : servo-moteur peut tourner, mais surtout maintenir sa position (tant couple le permet)

Fonctionnement : moteur toujours en action : tourne très peu dans un sens puis dans l'autre = garder équilibre

Principe : code=placer à une certaine position / potentiomètre qui tourne = récupère position rotation

🡪 compare à la demande => moteur ajuste position en envoyant ordre (son esclave = = ‘*asservissement*’)

**contraintes de rotation d'un servo-moteur** cas servo-moteurs amplitude 180° + ne contrôle pas la vitesse

**Connexion servo-moteur – Arduino attention en cas d’erreur, risque de griller le servo-moteur photoa\_4/a\_5**

3 fils : rouge=alimentation 5V / noir (marron)=ground / jaune (orange ou blanc)= commande sur pin 0 à 13

Remarque : si alimentation externe du moteur, relier ground Arduino, Alimentation, servo-moteur ensemble

**Envoyez des ordres à un servo-moteur : train d’impulsions électrique à intervalle et durée précis = ‘période’**

**photoa\_6 :** observer : valeur tension (+5V ou 0V) / période = 20 millisecondes / durée impulsion peut varier

**photoa\_7 :** 3 trains impulsions de période 20ms = 3 interprétation :

* + 1ms=SM position 0°/ 1.5ms=SM position 90°/ 2ms=SM position 180
  + map : int dureeImpulsion=map(angle,0,179,1000,2000); angle entre 0°-179°, résultat entre 1 000-2 000μs

Envoi : avec horloge interne de l'Arduino #delay() = delayMicroseconds(valeur);

= envoyer courant (HIGH ou LOW) dans PIN commande servo selon durée impulsion & intervalle impulsions

Code : incliner le servo-moteur à différents angles entre 0° et 179° sur pin 8

|  |
| --- |
| int periode=20000;// période entre chaque début d'impulsion en microsecondes  int pinServo=8; // variable pour le pin connecté à la commande du servo  void setup() {  pinMode(pinServo,OUTPUT);// on prépare le pin en mode OUTPUT  digitalWrite(pinServo,LOW); // on l'initialise à l'état bas  }  //boucle principale  void loop() {  for (int angle=0;angle<=180;angle+=20){//on fait varier l'angle de 0 à 180° par tranche de 20°  setAngle(angle);// on appelle la fonction setAngle définie plus bas  }  }  //fonction setAngle pour envoyer les impulsions  void setAngle(int a){  int duree=map(a,0,179,1000,2000);// on transforme l'angle en microsecondes et on stocke dans la variable duree  digitalWrite(pinServo,LOW);//on met le pin à l'état bas    // la boucle qui suit est nécessaire  // pour laisser le temps au servo d'aller à sa position  for (int t=0;t<300;t++){  digitalWrite(pinServo,HIGH);// on envoie l'impulsion  delayMicroseconds(duree); // pendant la bonne durée  digitalWrite(pinServo,LOW); // on stoppe l'impulsion  delayMicroseconds(periode-duree); // on attend le temps restant pour atteindre la période  }  } |

Résultat : position 0° puis par tranche de 20° jusqu'à 180° et revient à la position de départ et recommence

Pb potentiels :

* **ne revient pas à 0 =>** augmenter nb boucles du compteur  t dans setAngle() pour donner plus de temps
* **ne parcourt pas demi-tour complet** => changer valeurs mappage (ex : remplacer 1000-2000 par 500-2500)

= logique = trouver valeurs limites (haute et basse) pour ne jamais dépasser limites servo-moteur

* **bibliothèque Servo : fonctions qui maintiennent servo en position tout en exécutant une autre tâche**

include (directive pré-compilation) **:** #include <Servo.h> (en fait 2 fichier .c et .cpp)

classe : Servo monServo;

connexion pin : monServo.attach(8);

fonction : rotation d’un angle voulu : monServo.write(angle);

exemple1  : servo en position 0° puis 180° d'une seconde à la suivante

|  |
| --- |
| #include <Servo.h> //on importe la bibliothèque Servo  int pinServo=8; // variable pour stocker le pin pour la commande  Servo leServo; // on définit un objet Servo nommé leServo  void setup() {  leServo.attach(pinServo); // on relie l'objet au pin de commande  }  void loop() {  leServo.write(0); // on dit à l'objet de mettre le servo à 0°  delay(1000); // ce délai est nécessaire pour que le servo atteigne sa position  leServo.write(179); // position à 179, 180 est à éviter car cela forcerait le servo à dépasser ses limites  delay(1000); // attente à nouveau  } |

exemple2  : LED attachée au pin 13 clignote alors que le servo-moteur se déplace

|  |
| --- |
| #include <Servo.h> //on importe la bibliothèque Servo  int pinServo=8; // variable pour stocker le pin pour la commande  Servo leServo; // on définit un objet Servo nommé leServo  void setup() {  leServo.attach(pinServo); // on relie l'objet au pin de commande  pinMode(13,OUTPUT); //pin 13 en mode OUTPUT  }  void loop() {  leServo.write(0); // on dit à l'objet de mettre le servo à 0°  diode13(); // appel de la fonction diode13() définie plus bas  leServo.write(179); // position à 179°, 180° est à éviter  diode13(); // appel de la fonction  }  void diode13(){  // on fait clignoter 30 fois la LED 13  for (int t=0;t<30;t++){  digitalWrite(13,HIGH);  delay(100);  digitalWrite(13,LOW);  delay(100);  }  } |

Tests : varier valeurs angles, nb boucles et temps d'attente

**Potar et Servo**  : commander votre servo-moteur grâce au potentiomètre

Circuit : **photoa\_8**

|  |
| --- |
| #include <Servo.h>// on importe la bibliothèque  Servo servo; // on crée l'objet Servo  int pinServo=8; // on définit le pin lié à la commande du servo  int pinPotar=A0; // on définit le pin lié à la lecture du potenitomètre  void setup() {  servo.attach(pinServo); // on relie l'objet servo au pin de commande  }  void loop() {  int valeurPotar=analogRead(pinPotar); // lecture de la valeur du potentiomètre  int angle=map(valeurPotar,0,1023,0,179); // tranformation en angle  servo.write(angle); //mise en position du servo  } |

Remarque : impossible de connaître la position d’un servo de base (servo.read()  renvoie la dernière valeur)

=> impossible tester arrivée du servo à une position précise pour déclencher un évènement

**Accélerateur**  : servo pin8 + un bouton poussoir pin 2 en mode INPUT\_PULLUP

Principe : servo en 0°

* + poussoir appuyé (maintenu) passe à 45°-90°-135°-180° selon temps appui
  + poussoir relâché "redescend" par les positions inverses, avec les mêmes paliers de temps, jusqu'à 0°
  + si on appuie durant la "descente" le servo "remonte".

Difficulté : méthode pour calculer la durée d’appui ou relachement

Circuit : **photoa\_9**

Programme : 1/ sans palier 2/ avec paliers d’angles

* **Accélérateur sans paliers d’angles** se déplace tant qu'on appuie, et revient lorsqu'on relâche

Principe : objet Servo pin 8 / poussoir pin 2 mode INPUT\_PULLUP / cumul= temps appui / map temps-angle

+ limites pour le temps d'appui

|  |
| --- |
| #include <Servo.h> //import de la bibliothèque Servo  Servo accel; //création de l'objet Servo "accel"  int pinServo=8; //pin de commande du servo  int pinBouton=2;//pin de lecture du bouton poussoir  int cumul=0; //variable d'appui  void setup() {  pinMode(pinBouton,INPUT\_PULLUP); //mode INPUT\_PULLUP pour le poussoir  accel.attach(pinServo); //liaison de l'objet Servo au pin de commande  Serial.begin(9600);//pour lecture sur la console (Optionnel)  }  void loop() {  boolean etatBouton=digitalRead(pinBouton); //lecture de l'état du bouton    //si le bouton est appuyé  if (!etatBouton){// en mode INPUT\_PULLUP on obtien 0 quand on appuie !  cumul++; // on fait augmenter la valeur de la variable  if (cumul>1000) //test limite d'augmentation  cumul=1000;//mise à limite si dépassement  }    //si le bouton n'est pas appuyé  else{  cumul--; //on fait diminuer la valeur de la variable  if (cumul<0) //test si limite de diminution  cumul=0;//mise à la limite si dépassement  }  Serial.println(cumul); //on affiche la valeur sur la console (Optionnel)  int angle=map(cumul,0,1000,0,179); //on transforme en angle  accel.write(angle); //on positionne le servo  } |

Rappel : if(!contact) = if(contact==0)

Langage : switch(valeur) :

|  |  |
| --- | --- |
| int nombreDePersonnes=5;  int nombreDeChaises=3;  if (nombreDeChaises==nombreDePersonnes){  Serial.print("Tout le monde peut s'asseoir");  }  else if (nombreDePersonne-nombreDeChaises==1){  Serial.print("Il y en a 1 qui reste debout");  }  else if (nombreDePersonne-nombreDeChaises>1 && nombreDeChaise!=0){  Serial.print("Il y en a plusieurs qui restent debout");  }  else {  Serial.print("Tout le monde reste debout");  } | int nombreDePersonnes=5;  int nombreDeChaises=3;  int diff=nombreDePersonnes-nombreDeChaises;  switch (diff){  case 0:  Serial.print("Tout le monde peut s'asseoir");  break;  case 5:  Serial.print("Tout le monde reste debout");  break;  case 1:  Serial.print("Il y en a 1 qui reste debout");  break;  default:  Serial.print("Il y en a plusieurs qui restent debout");  } |

* **Accélérateur avec paliers d’angles**

Principe : utiliser la fonction switch pour définir les paliers d'arrêt pour le servo

|  |
| --- |
| #include <Servo.h> //import de la bibliothèque Servo  Servo accel; //création de l'objet Servo "accel"  int pinServo=8; //pin de commande du servo  int pinBouton=2;//pin de lecture du bouton poussoir  int cumul=0; //variable d'appui  void setup() {  pinMode(pinBouton,INPUT\_PULLUP); //mode INPUT\_PULLUP pour le poussoir  accel.attach(pinServo); //liaison de l'objet Servo au pin de commande  Serial.begin(9600);//pour lecture sur la console (Optionnel)  }  void loop() {  boolean etatBouton=digitalRead(pinBouton); //lecture de l'état du bouton    //si le bouton est appuyé  if (!etatBouton){// en mode INPUT\_PULLUP on obtien 0 quand on appuie !  cumul++; // on fait augmenter la valeur de la variable  if (cumul>1000) //test limite d'augmentation  cumul=1000;//mise à limite si dépassement  }    //si le bouton n'est pas appuyé  else{  cumul--; //on fait diminuer la valeur de la variable  if (cumul<0) //test si limite de diminution  cumul=0;//mise à la limite si dépassement  }    Serial.println(cumul); //on affiche la valeur sur la console (Optionnel)  int pos=map(cumul,0,1000,0,4);//on mappe de 0 à 4 (5 positions) dans un variable pos  int angle=0; //on initialise une variable angle    switch (pos){// va switcher en fonction de la valeur pos  case 1: angle=45; break;  case 2: angle=90; break;  case 3: angle=135; break;  case 4: angle=179; break;  default: angle=0;  }  accel.write(angle); //on place le servo à la position angle  } |

**13\_ CAPTEURS ELECTRONIQUES**

**matériel requis**

Capteur de contact / Tilt sensor / Photorésistance / Photodiode infrarouge (et LED infrarouge)

Récepteur infrarouge 38 KHz (TSOP38238) / Capteur de température (TMP36) /Capteur à ultrasons (SRF05)

**types de capteurs**

* **capteurs tout-ou-rien :** que 2 données (souvent HIGH et LOW)
  + lecture : sur ports numériques et stocke souvent dans une variable de type booléenne
  + montage : résistance pull-up ou pull-down pour éviter valeurs erratiques (ou pull-up interne)
* **capteurs de variation :** varier de tension de sortie soit en résistant, soit en produisant un courant.
  + lecture : sur ports analogiques = transforme tension récupérée ( 0V-et 5V) en entier entre 0 et 1024.
  + montage : résistance comme pont diviseur de tension (voir schéma)
  + utile de réaliser une sorte d'étalonnage pour évaluer les limites de variation de tension liée au capteur. Je vous fournis un [programme](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/les-capteurs-electroniques#/id/r-3236693) dans ce chapitre qui peut vous y aider, mais vous concevrez vite le vôtre,
* **capteur de contact photod\_1** sorte bouton poussoir qui s'actionne grâce à la force exercée sur un petit levier

capteur tout ou rien : aussi appelés capteurs de collision, microrupteurs (petits + laisser passer ou non courant)

montage : assez faciles à fixer mais prévoir soudure pour relier les pattes au montage : **photod\_2**

* **2 pattes = interrupteur simple** :
  + soit ouvert => se ferme au contact (type "NO" pour *Normally Open*)
  + soit fermé => s'ouvre au contact (type "NC" pour *Normally Closed*)
* **3 pattes = un inverseur** : laisse passer le courant vers l'une ou l'autre des pattes

connexion

* 2 pattes : comme pour un bouton poussoir (résistance en mode pull-up/pull-down ou INPUT\_PULLUP)
* 3 pattes (inverseur) : patte COM=pin de lecture / patte NC=ground / patte NO=+5V **photod\_3**

programme comme un bouton poussoir = récupère valeur haute (HIGH ou 1) ou basse (LOW ou 0)

|  |
| --- |
| int pinContact=7; // pin de lecture du contact  void setup() {  Serial.begin(9600); // initialisation de la connexion série  pinMode(pinContact,INPUT\_PULLUP); //active la résistance pull-up interne  }  void loop() {  boolean etatContact=digitalRead(pinContact);  if (!etatContact) //test inverse car mode INPUT\_PULLUP  Serial.println("Contact");  else  Serial.println("Pas de contact");  } |

utilisation : pour vérifier si robot touche obstacle / portail en fin de course / contact entre 2 objets mobiles...

* **tiltsensor photod\_5** accéléromètre du pauvre : bille métal fait contact sur pièces métalliques si le mvt le permet

capteur tout-ou-rien : mouvement doit être brusque ou changement d’inclinaison

connexion : comme capteur de contact

programme : comme capteur de contact

utilisation :

* percevoir un mouvement brusque (chute ou choc) car on capte alors une absence de contact.
* lire grossièrement inclinaison avec 2 tiltsensors positionnés en un V = inclinaison excessive coupe contact
* **photoresistance photod\_6** capteur de lumière : plus il y a de lumière, plus la résistance est basse

capteur de variation => connecter à un pin analogique

circuit : **photod\_7**

programme : +photorésistance  reçoit de la lumière, + résistance est faible, + valeur affichée élevée

|  |
| --- |
| int pinPR=A0; //pin de connexion pour la photorésistance  void setup() {  Serial.begin(9600);  }  void loop() {  int valeur=analogRead(pinPR); // on lit la valeur transmise par la photorésistance  Serial.println(valeur); // on l'affiche  } |

programme affiche valeurs => permet ajuster entre 0% et 100% (bien d’étalonner un lieu)

|  |
| --- |
| int pinPR=A0; //pin de connexion pour la photorésistance  int valMin=1024; // on initialise la valeur minimale au plus haut  int valMax=0; // et la valeur maximale au plus bas  void setup() {  Serial.begin(9600);  }  void loop() {  int valeur=analogRead(pinPR); // on lit la valeur transmise par la photorésistance  if (valeur>valMax) //on compare avec valMax  valMax=valeur; // on modifie valMax  if (valeur<valMin) // on compare avec valMin  valMin=valeur; // on modifie valMin  int pourcentage=map(valeur,valMin,valMax,0,100); //pourcentage entre les bornes  //Séquence d'affichage  Serial.print("Valeur : ");  Serial.print(valMin);  Serial.print(" < ");  Serial.print(valeur);  Serial.print(" < ");  Serial.print(valMax);  Serial.print(" soit : ");  Serial.print(pourcentage);  Serial.println(" %");  } |

utilisation : déclencher un évènement en fonction de la luminosité : pièce qui s'éclaire lorsqu'on entre, inclinaison stores pour gérer entrée de lumière, guidage d'un robot grâce à une source de lumière, allumage d'une lampe si trop d'obscurité...

remarque : si la zone trop éclairée, réduire la résistance de 10KΩ à 1KΩ ou utiliser un potentiomètre

* **photodiode infrarouge photod\_a** réagit à la lumière infrarouge en produisant un signal électrique
* aussi des photodiodes rectangulaires avec un angle coupé qui représente la borne
* souvent avec une coque de plastique noir pour filtrer la lumière visible et capter que lumière infrarouge

ondes : rayons gamma / rayons X / UV / lumière visible / IR / micro-ondes / ondes radio.

= invisible pour l'œil humain mais on sait l'émettre et la capter (exp : télécommandes)

capteur de variation => connecter à un pin analogique

matériel : LED infrarouge en complément (à protéger par une résistance) **photod\_8, photod\_9**

**prévoir de l’installer sur une autre plaque pour pouvoir l’éloigner et faire varier les valeurs**

**Montage dans le sens "normal"**

Montage : **photod\_b**

* patte courte=ground / patte longue=+5V, lecture côté patte courte / résistance importante (320KΩ)
* LED IR comme une LED normale et faisceau dirigé vers la photodiode.

|  |
| --- |
| void setup() {  Serial.begin(9600);  }  void loop() {  Serial.println(analogRead(A0)); //lecture du CAN A0 connecté à la patte - de la photodiode  } |

Test : en passant un objet opaque entre la LED IR et la photodiode,varie entre 30 et 800 environ.

**Montage "à l'envers"** : **photod\_c**

Principe : photodiode émet un courant lorsqu'elle reçoit un faisceau d'ondes infrarouges.

Montage : diode dans le mauvais sens= petite patte +5V, grande ground / résistance plus faible (ici 1KΩ)

Programme : même programme

Résultat : même plage de lecture entre 30 et 880

Explication : longueur d'onde diode fixe, tension varie (résistance +- importante) = quantité IR lue par CAN

= liée à la distance entre la diode et la photodiode

🡪 indique une notion de distance mais pas transformable en cm car trop variable en fonction du montage.

Utilisation : barrière infrarouge / diriger robot mobile vers une source IR : +réception forte, +direction bonne

* **récepteur infrarouge photod\_d** exemple de récepteur infrarouge : le TSOP38238 [datasheet du TSOP38238](http://www.vishay.com/docs/82491/tsop382.pdf)

Différences avec photodiode : de type **tout-ou-rien** + **seulement sous forme de signal à 38KHz** environ (=26ms)

=> surtout capteur de télécommande infrarouge (ou tout ce qui émet IR à la bonne fréquence, autre Arduino)

Complément : chapitre complet dans le cours de perfectionnement « télécommande »

Connexions : **photod\_e** 3 pattes : 1(OUT) pin numérique / 2 ground / 3 +5V

|  |
| --- |
| void setup() {  Serial.begin(9600);  }  void loop() {  Serial.println(digitalRead(7));  } |

Résultat : rien avec LED IR mais avec télécommande IR (ex télé) vers le capteur = succession 0 & 1 sur console

|  |
| --- |
| void setup() {  Serial.begin(9600);  pinMode(7, INPUT);  }  void loop() {  int p = 0; //variable de comptage pour tableau  int tLow,tHigh; //variables de stockage de temps fictif  int duree[32][2]; //tableau de stockage des temps de 32 sur 2  Serial.println("Go"); //lancement de la procédure  while (digitalRead(7)) {} // attente de changement d'état    while (p < 32) { // boucle de stockage des données  tLow=0; //initialisation du temps fictif pour état BAS  tHigh=0;//initialisation du temps fictif pour état HAUT  while (!digitalRead(7)) { //tant que le récepteur reçoit un signal LOW  tLow++; //on incrémente le temps fictif pour état bas  }  while (digitalRead(7)) { //tant que le récepteur reçoit un signal HIGH  tHigh++; //on incrémente le temps fictif pour état haut  }  duree[p][0] = tLow; //stockage dans tableau  duree[p][1] = tHigh; //stockage dans tableau  p++;  }    //affichage du tableau  Serial.print("LOW\t");  for (int p = 0; p < 32; p++) {  Serial.print(duree[p][0]);  Serial.print("\t");  }  Serial.println();  Serial.print("HIGH\t");  for (int p = 0; p < 32; p++) {  Serial.print(duree[p][1]);  Serial.print("\t");  }  Serial.println();    delay(3000);// attente pour nouvelle procédure  } |

Résultat : approximation temps où le signal est aux états haut et bas mais pas data utilisables

= fonction digitalRead()  trop lente pour lire correctement le pin 7 (voir cours de perfectionnement)

Utilisation : fait souvent partie des kits du commerce mais utilisation assez complexe (envoi-réception code IR)

* **Capteur de température photod\_f** exemple TMP36 (températures entre -50°C et 125°C) [datasheet](http://www.analog.com/media/en/technical-documentation/data-sheets/TMP35_36_37.pdf)

ressemble beaucoup à un transistor # 3 pattes, côté plat, côté rond ... mais inscriptions dessus = nom

Connexions : pattes en bas & côté plat face : 3 pattes : gauche=+5V / droite=ground / centre= un CAN

(si erreur chauffe, brûle les doigts si on touche, peu griller)

Data : **photod\_g** rapport entre la tension lue à la sortie (patte du centre) et la température

Rappel : CAN convertissent tension lue entre 0V et 5V en nombre entre 0 et 1023. (ok pour 5V en entrée)

Programme : affiche la température lue par le TMP36

|  |
| --- |
| void setup() {  Serial.begin(9600);  }  void loop() {  int valeur=analogRead(A0);//on lit la valeur au CAN 0  int t=map(valeur,0,1023,0,5000);//on transforme la valeur lue en valeur de tension entre 0 et 5000 mV (car pas de virgule)  int tmp=map(t,0,1750,-50,125); //on transformela tension (de 0 à 1750mV en température (de -50°C à 125°C);  Serial.println(tmp); //on affiche le résultat  } |

Utilisation : station météo, déclenchement de chauffage, variation température 1 liquide …

* **Capteur ultrasons photod\_h** SRF05 capte > 20KHz (précisément de 20 KHz à 10 MHz [Notice du SRF05](http://www.robot-electronics.co.uk/htm/srf05tech.htm)

spectre audible : entre 20Hz et 20KHz

rappel : vitesse **lumière** : **299 792 458 m/s** **dans le vide** / vitesse son : selon matière : **son dans l'air** **340  m/s)**

notice : mesure échos à distance entre 3cm et 4m / mesure toutes les 50ms / cône écoute 30° (-15°,+15°)

connexion : 1/ 1 patte émission & 1 patte réception ou 2/ 1 seule patte émission-réception (on fait ça)

circuit : cas 2 donc 2 pattes connectées au ground

Principe :

1. Arduino (prog) envoie signal haut (HIGH) pendant 10μs … attend le retour en comptant temps passé (en μs)
2. SRF05 envoie un "beep" ultrasons (à la fréquence de 40KHz) et attend l'écho (=rebond sur obstacle)
3. Quand son capté en retour, informe Arduino en lui renvoyant un signal HAUT.
4. Arduino (prog) exploite données du temps entre envoi et réception.

Calcul :

* transformer temps en distance ? 340ms/s = 34000cl/ 1000000μs donc pour 1cm= 1000000\*1/34000=29 µs
* parcourt la même distance à l'aller et au retour = diviser temps en µs par 58 et on obtient => distance

Programme : faire une moyenne sur 10 mesures pour stabiliser le résultat

|  |
| --- |
| int pinSRF = 7; //pin digital pour l'envoi et la réception des signaux  int vSon=59; //valeur de temps en µs d'un aller retour du son sur 1cm  void setup() {  Serial.begin(9600); //on initialise la communication série  }  //boucle principale  void loop() {  int distance=mesureDistance(); //on récupère la valeur de distance grâce à la fonction créee plus bas  Serial.println(distance); // on affiche la distance en cm  }  //fonction de mesure de distance avec SRF05  int mesureDistance() {  unsigned long mesure = 0; // variable de mesure  unsigned long cumul = 0; //variable pour la moyenne  for (int t = 0; t < 10; t++) { // boucle pour effectuer 10 mesures  pinMode (pinSRF, OUTPUT); //on prépare le pin pour envoyer le signal  digitalWrite(pinSRF, LOW); //on commence à l'état bas  delayMicroseconds(2); //on attend que le signal soit clair  digitalWrite(pinSRF, HIGH);//mise à l'état haut  delayMicroseconds(10); //pendant 10 µs  digitalWrite(pinSRF, LOW); //mise à l'état bas  pinMode(pinSRF, INPUT); //on prépare le pin pour recevoir un état  mesure = pulseIn(pinSRF, HIGH); // fonction pulseIn qui attend un état haut et renvoie le temps d'attente  cumul+=mesure; //on cumule les mesures  delay(50); //attente obligatoire entre deux mesures  }  mesure=cumul/10; //on calcule la moyenne des mesures  mesure=mesure/vSon;//on transforme en cm  return mesure; //on renvoie la mesure au programme principal  } |

Remarques :

* unsigned long  entiers entre 0 et 4 294 967 295 : +sûr si nombres augmentent rapidement sinon erreur

(dépassement mémoire = modifie nb en négatif (signé) ou à 0 (non signé) => fausse valeurs (tester avec int)

* pulseIn(pin,ETAT) : compte temps en μs entre appel et pin passe à l'état indiqué (HIGH) ou (LOW)
* delayMicroseconds() = permet d'attendre en microsecondes !

Utilisation :

* souvent détection obstacle à distance par robots mobiles (vitesse/ arrêter / chercher passage / repartir)
* appareil de mesure de volume (stockage données de distance sur 3 dimensions).
* stabilisateur d'altitude (4m maximum) pour un objet volant.

**1\_ DEFINITIONS ARDUINO** (société italienne Smart Projects)

**1\_ DEFINITIONS ARDUINO** (société italienne Smart Projects)