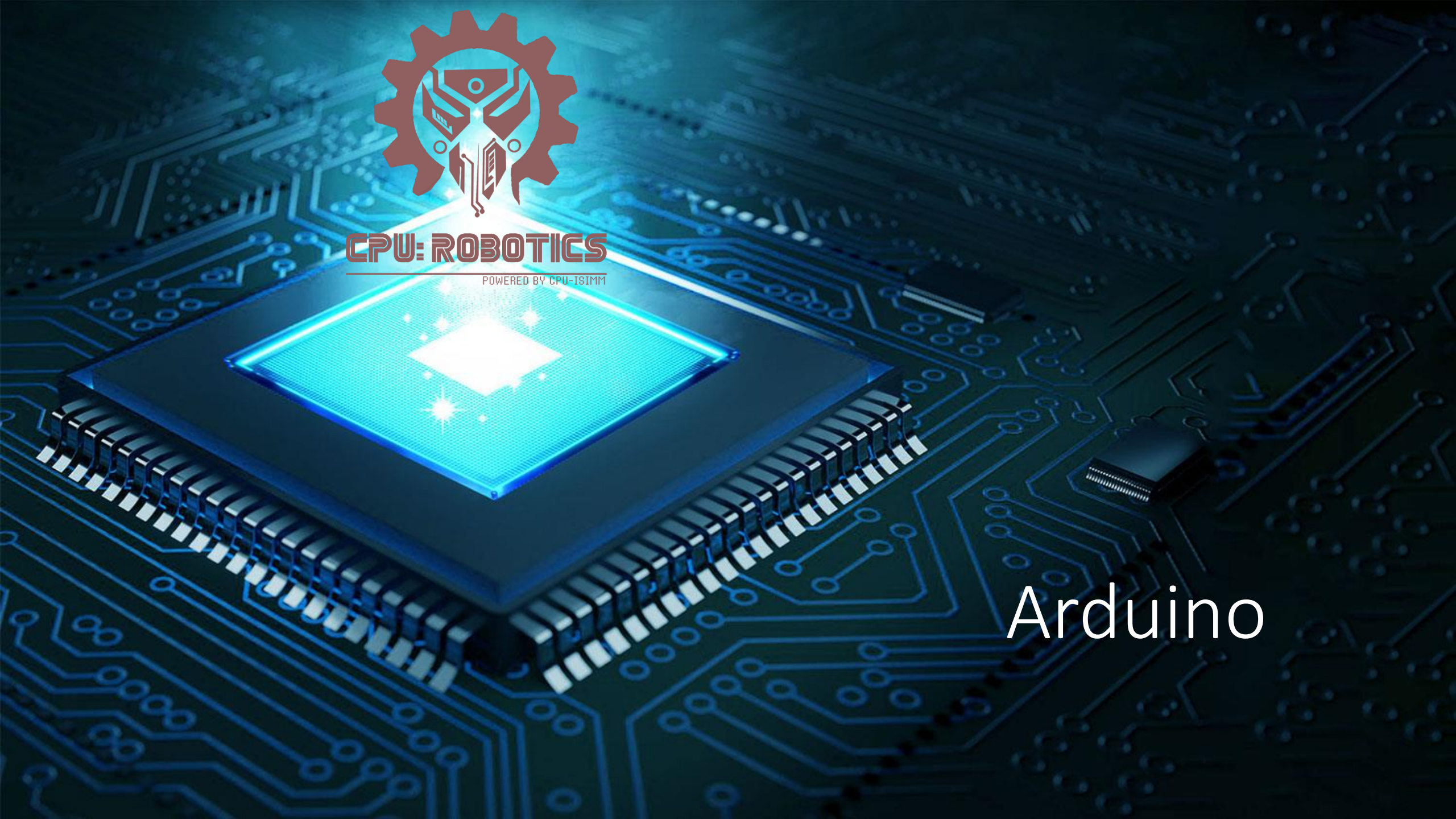
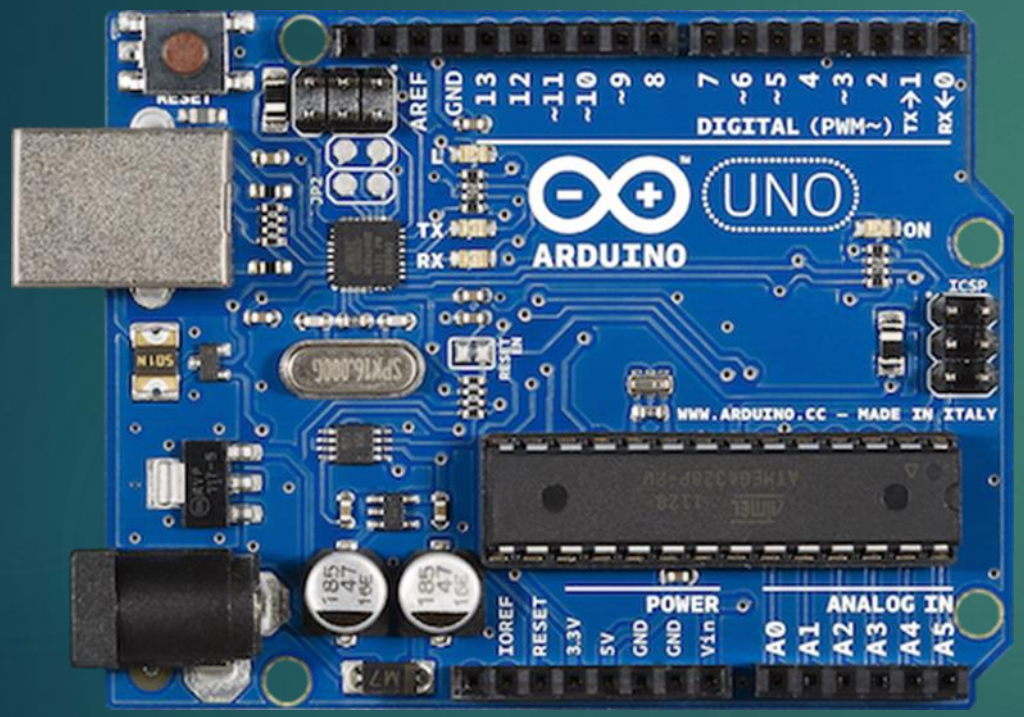


Arduino

Instructeur : yasser jamli

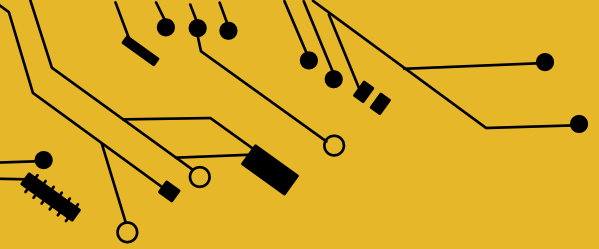
Introduction

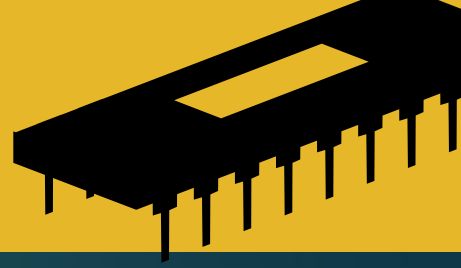


quelques exemples de ce que vous pouvez réaliser avec une telle carte :
-Un robot mobile capable d'éviter les obstacles ou de suivre une ligne au sol ;
-Une interface entre votre téléphone mobile et les éclairages de votre maison ;
-Des afficheurs d'informations à base de textes défilants sur des panneaux à LEDs ;
-Une station météorologique consultable sur le Web ;
-Un pilote de caméra de surveillance.
De plus en plus utilisé dans le monde de l'éducation, l'Arduino est aussi apprécié dans le cadre des technologies embarquées, du modélisme et peut rapidement devenir un hobby. La force de cette carte Arduino à microcontrôleur est, au-delà de son prix, la facilité avec laquelle on peut réaliser des petits projets viables, et surtout la communauté libre qui s'est développée autour d'elle.

Qu'est-ce que l'Arduino ?

Arduino est un **circuit imprimé** en **matériel libre** sur lequel se trouve un **microcontrôleur** qui peut être programmé pour analyser et produire des **signaux électriques**.

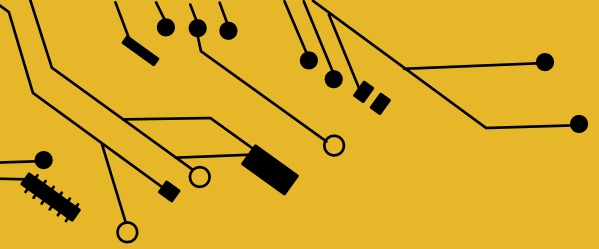


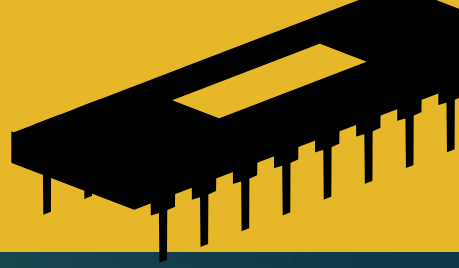


Circuit imprimé : C'est une sorte de plaque sur laquelle sont soudés plusieurs composants électroniques reliés entre eux par un circuit électrique plus ou moins compliqué. L'Arduino est donc un circuit imprimé. La photo donne une idée de la taille par rapport à la connexion USB carrée (à gauche sur la photographie, la même que sur votre imprimante par exemple).

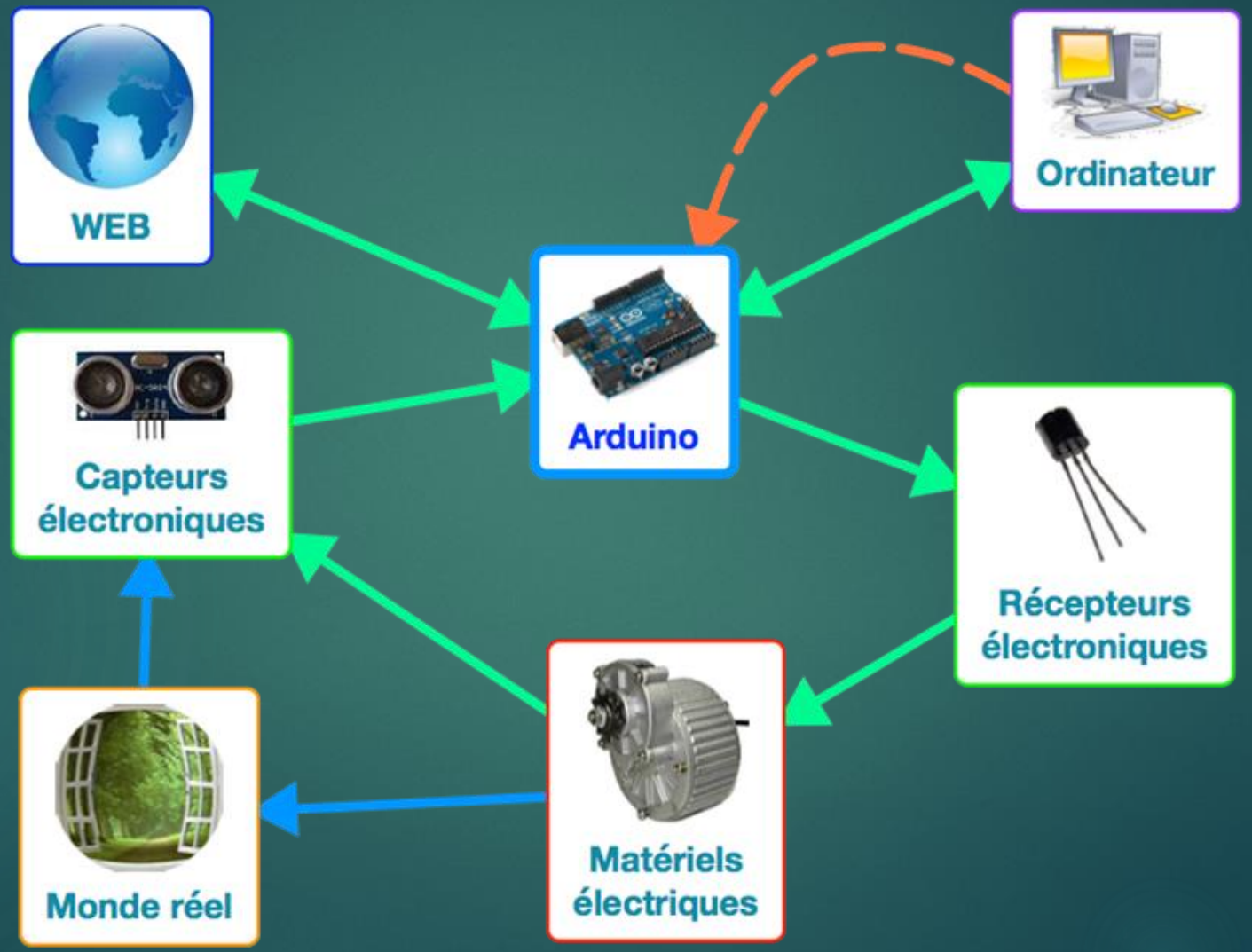
Matériel libre : En fait, les plans de la carte elle-même sont accessibles par tout le monde, gratuitement. La notion de libre est importante pour des questions de droits de propriété.

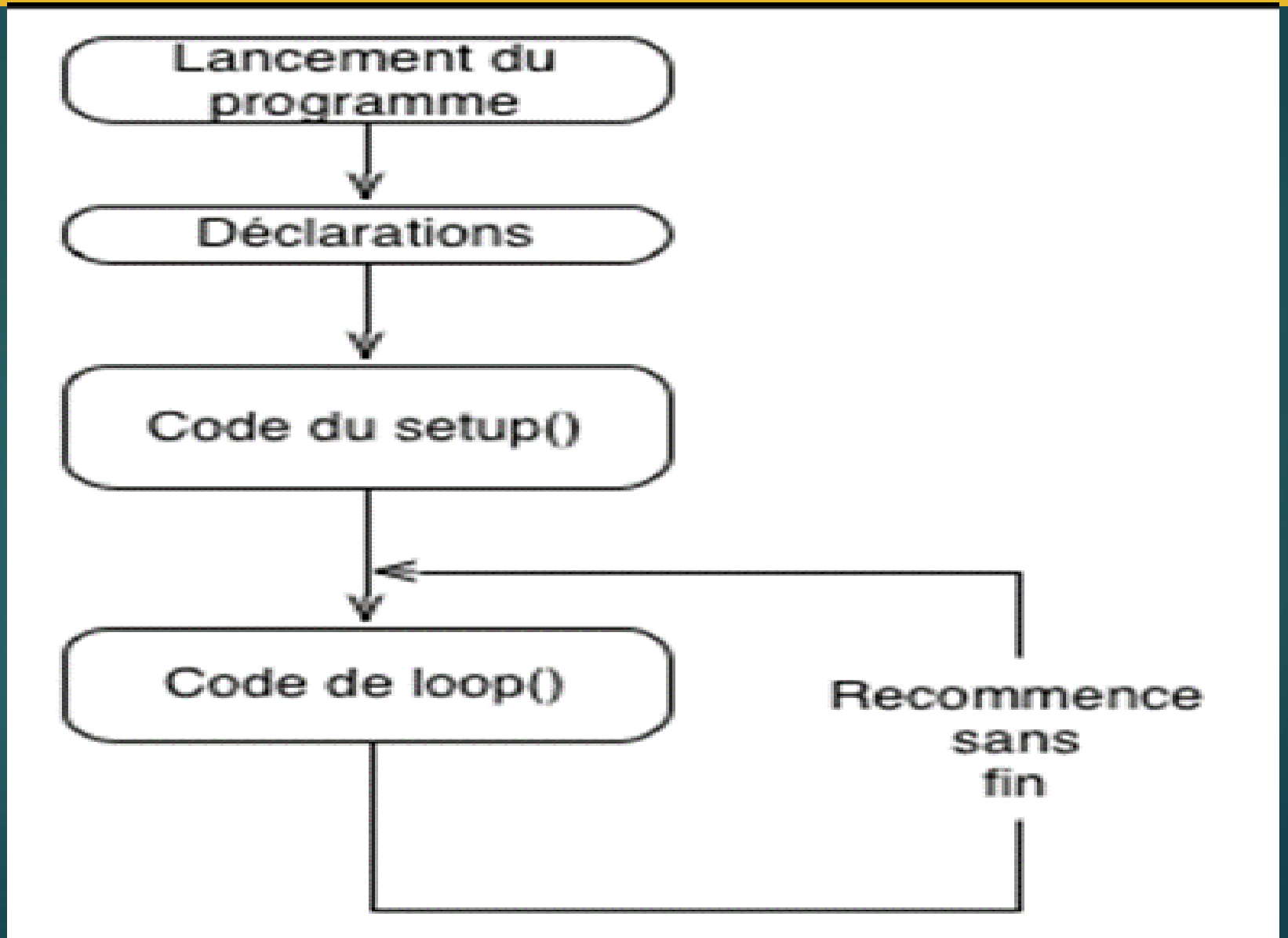
Microcontrôleur : C'est le cœur de la carte Arduino. C'est une sorte d'ordinateur minuscule (mémoire morte, mémoire vive, processeur et entrées/sorties) et c'est lui que nous allons programmer. Sur la photo précédente, c'est le grand truc rectangulaire noir avec plein de pattes. Une fois lancé et alimenté en énergie, il est autonome. La force de l'Arduino est de nous proposer le microcontrôleur, les entrées/sorties, la connectique et l'alimentation sur une seule carte. La carte Arduino est construite autour d'un microcontrôleur Atmel AVR (pas toujours le même en fonction de la date de sortie de la carte) avec une capacité de mémoire de 32000 octets pour l'Arduino UNO. Soit 32 Ko, ce qui n'est vraiment pas beaucoup et qui permet pourtant de réaliser un max de projets !

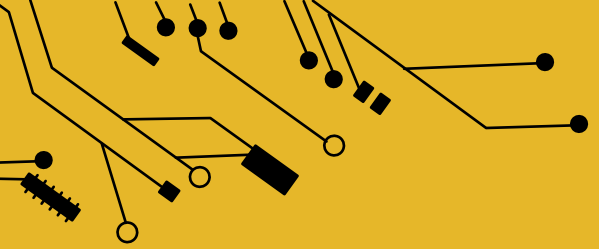


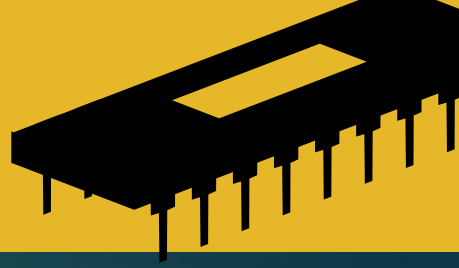


Pour ceux qui se demandent pourquoi ces cartes Arduino ont des noms aux consonances italiennes... et bien la réponse se trouve dans leur origine de fabrication : c'est la société italienne Smart Projects qui crée les modules de base des cartes Arduino ! À noter que quelques cartes sont également fabriquées par SparkFun Electronics, une société américaine.
L'Arduino est donc une carte qui se connecte sur l'ordinateur pour être programmée, et qui peut ensuite fonctionner seule si elle est alimentée en énergie. Elle permet de recevoir des informations et d'en transmettre depuis ou vers des matériels électroniques : diodes, potentiomètres, récepteurs, servo-moteurs, moteurs, détecteurs... Nous en ferons une description plus précise au fur et à mesure de ce cours.


WEB
Ordinateur
Arduino
Capteurs électroniques
Récepteurs électroniques
Monde réel
Matériels électriques


Lancement du programme
Déclarations
Code du setup()
Code de loop()
Recommence sans fin





<u>Vous avez appris les mots-clés :</u>

- setup() qui s'exécute qu'une fois ;
- Loop() qui s'exécute à l'infini ;
- void (qui se met toujours devant loop et setup et dont vous comprendrez le sens plus loin dans ce cours) ;
- pinMode() qui permet de définir une sortie en mode envoi d'électricité ou non ;
- digitalWrite() qui envoie de l'électricité par une sortie ;
- delay() qui met le programme en pause pendant un nombre défini de millisecondes.

Vous savez aussi que les **accolades** délimitent des blocs d'instructions et que le **point-virgule** termine une instruction.

Utilisez les constantes, les variables, les conditions et le moniteur série

Constantes :
On écrit le nom des constantes en majuscules (ça fait partie des conventions de programmation) ça permet de savoir que ce sont justement des constantes.
Le nom de notre constante ne s'écrit pas en bleu car elle n'est pas une constante réservée à l'IDE mais bien une à nous, qu'on a fait tout seul comme un grand !
La déclaration d'une constante (c'est-à-dire sa construction) commence par le mot clé **const** suivi du type de constante (dans notre cas la constante CONNEXION est de type "int", qui signifie nombre entier - nous y reviendrons plus loin).
La déclaration d'une constante est toujours de la forme :

const type NOMDELACONSTANTE=valeur;

variables

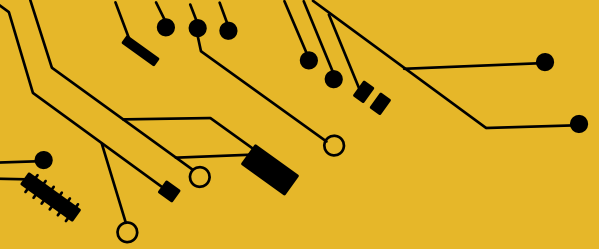
La déclaration de variable peut se faire de différentes façons. Pour des raisons de bonnes habitudes et de lisibilité, nous allons utiliser une méthode en deux temps. Je la trouve plus utile dans le cas où vous voudriez un jour programmer sur autre chose que l'Arduino et en plus elle est moins coûteuse en espace mémoire (ce qui, vous le verrez est important pour les microcontrôleurs).
Premier temps : je dis à l'IDE que je réserve une place en mémoire pour une variable et je lui donne un nom (cette opération est faite une seule fois dans le programme).
Autres temps : je dis à l'IDE d'affecter une valeur à la variable (cette opération peut être faite à tout moment dans le programme).

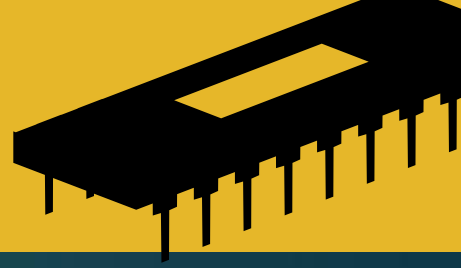
Dans le cas de variables, l'IDE ne procède pas comme pour les constantes. Lors de la déclaration, il va traduire en langage machine qu'il faut réserver une place de la bonne dimension (c'est le type) dans la mémoire vive (c'est-à-dire modifiable) de l'Arduino. Puis, lors des modifications de variables, l'Arduino ira déposer dans sa mémoire, à la place prévue, la nouvelle valeur, qui à chaque fois remplacera la précédente.
Tant qu'aucune valeur n'est attribuée à la variable, la case mémoire contient n'importe quoi. Si vous utilisez une variable non initialisée, vous obtiendrez des résultats souvent étonnants qui risqueront de planter votre programme.
Pour initialiser une variable, on peut utiliser la boucle du setup() ou un autre endroit dans le programme. L'important c'est que la variable soit initialisée avant son utilisation.

Pour les variables il existe une façon agréable, lisible et surtout conventionnelle de les nommer : on écrit le nom tout attaché et on met des majuscules à chaque mot sauf au premier.

Le typage des variables et constantes

Le **typage** est obligatoire lorsqu'on définit une variable ou une constante. Dans certains langages (comme le Python par exemple), il n'est pas nécessaire car le compilateur se chargera de le faire et l'exécution du programme s'occupera des modifications éventuelles. Mais pour l'Arduino (et le langage C) on n'a pas le choix.
Chaque type de variable ne prend pas le même espace en mémoire et ne sera pas traité de façon équivalente lors de l'exécution. Je ne vais pas vous faire un cours pesant sur la mémoire des processeurs (bien des cours sur ce site le font très bien) mais je vais vous en faire un survol.





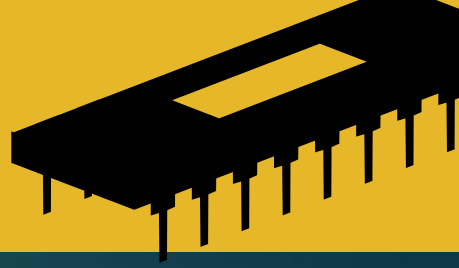
type	taille en byte	valeurs stockées
boolean	1	true ou false
char	1	un caractère ou un entier entre -128 et 127
unsigned char	1	un entier entre 0 et 255
byte	1	un entier entre 0 et 255
int	2	un entier entre -32768 et 32767
unsigned int	2	un entier entre 0 et 65535
float	4	un décimal, précis à 7 chiffres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

Portée des variables ou "scope"

Rappelez-vous que les accolades définissent des blocs de code à exécuter. Et bien si vous définissez une variable dans un bloc de code (donc entre des accolades), elle ne pourra être appelée ou modifiée QUE dans CE bloc de code.
Pour le moment vous ne connaissez que 2 types de blocs de code :
setup() (qui s'exécute une fois), et
Loop() (qui s'exécute à l'infini).
Vous verrez qu'il en existe bien d'autres.
Une fois que l'on sort des accolades du bloc, la place utilisée par la variable en mémoire est libérée. Ce qui nous laisse la possibilité de l'utiliser pour une autre variable.

Une variable globale est accessible (et modifiable) depuis n'importe quel endroit du programme. C'est pratique, mais ça nécessite de faire attention à son nom pour bien la repérer et surtout, une variable globale ne libère jamais sa place occupée en mémoire. L'utilisation excessive de variables globales peut parfois gêner l'exécution du programme en particulier pour l'Arduino dont la place mémoire est très restreinte.

Le moniteur série



Comme vous l'avez sûrement remarqué, l'Arduino de base n'a pas d'écran. On peut lui en ajouter un, nous aurons l'occasion de le voir plus tard, mais en attendant, on ne peut rien afficher. C'est là qu'entre en scène le **moniteur série**, qui est une fenêtre que l'on peut ouvrir en cliquant sur la loupe du logiciel Arduino. Cette fenêtre reçoit et envoie des infos sous forme de séries de bytes aux ports concernés. Donc grâce au moniteur série, l'Arduino peut (à condition d'être connecté à un PC) envoyer des infos à l'ordinateur qui va pouvoir les afficher en temps réel.
Pour ce faire on fait appel à une bibliothèque spéciale, déjà incluse dans l'IDE : la bibliothèque **Serial.**
Une **bibliothèque** est un fichier dans lequel sont stockés des sous-programmes, des variables et des constantes. Une fois ajouté dans notre programme, on a le droit d'utiliser leur fonctionnalité et leurs types de données. On utilise pour ce faire les mots-clés liés à la **bibliothèque** utilisée.
Pour utiliser la possibilité de communication entre l'ordinateur et l'Arduino, on procède en deux étapes :
On initialise la communication (ça se fait dans le setup()).
On communique (ça se fait souvent dans la loop() mais possible dans le setup()).

La LED TX montre une transmission envoyée par l'Arduino, la LED RX montre une transmission reçue par l'Arduino.
Pour comprendre le principe d'utilisation d'une bibliothèque :
Les codes liés à une bibliothèque commencent tous par son nom (ici Serial) suivi d'un point et de la commande.
La commande est spécifique à la bibliothèque utilisée.
La liste des commandes liées à une bibliothèque est ce qu'on nomme la documentation. Elle est souvent liée à la bibliothèque.

Le 9600 dans les parenthèse de l'initialisation correspond à un nombre de caractères par seconde qu'on appelle des bauds. L'Arduino peut donc envoyer un maximum de 9600 caractères par seconde à l'ordinateur dans cette configuration. Ne pas confondre avec l'unité bps, qui veut dire bits par seconde. Un caractère pour l'Arduino c'est 8 bits.

Code	Condition testée
==	égal
>=	supérieur ou égal
<=	inférieur ou égal
>	supérieur
<	inférieur
!=	différent (non égal)

Programme "Arduino compte seul

L'objectif de ce programme est de faire compter l'Arduino tout seul. Je m'explique :
Tout d'abord, l'Arduino doit afficher sur le moniteur série le nombre 1 et allumer une fois la LED 13.
Puis, après un temps d'arrêt d'une seconde, il doit afficher le nombre 2 et faire clignoter 2 fois la LED 13 à 2Hz.

Encore un arrêt d'une seconde, puis affichage du chiffre 3 avec 3 clignotements (toujours à 2 HZ).
L'opération se répète jusqu'au nombre 20, puis l'Arduino envoie « Ayé !" sur le moniteur série et plus rien ne se passe.

TP : Programme Table de multiplication

Dans ce programme, je vais vous demander d'afficher la table de multiplication par 7 des nombres de 0 à 14. Voici quelques règles à respecter :

Le programme devra afficher la table de multiplication une seule fois, il faudra donc utiliser une condition pour vérifier si elle a été affichée.

Pour rendre le programme facile à modifier, vous stockerez le nombre 7 dans une variable de type int. Nous apprendrons plus tard comment envoyer une information à l'Arduino depuis le moniteur, ce qui nous permettra d'afficher la table de notre choix.)

Le moniteur devra afficher une table qui ressemble à ça :

```
**************************
Table de multiplication
La table de : 7

0 x 7 = 0
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
11 x 7 = 77
12 x 7 = 84
13 x 7 = 91
14 x 7 = 98

**************************
```

Qu'est-ce qu'un tableau en programmation ?

C'est une façon de stocker des variables dans un même endroit au lieu de les créer séparément et leur donner des noms différents.
Les tableaux ne peuvent stocker **que des variables de même type** !

Comment déclarer un tableau ?

-*int* pinLed[3]; // ménage un espace dans ta mémoire pour stocker pour un tableau de type "int" avec le nom générique pinLed qui contiendra 3 valeurs.

-*int* pinLed[3]={2,4,6};

```
-int pinLed[3];     // déclaration du tableau
pinLed[0]=2;      // on donne la valeur 2 à la première valeur
pinLed[1]=4;    // 4 à la deuxième
pinLed[2]=6; // 6 à la troisième
```

Projet 1 : “Blink à trois”

Voici la description du programme :
Les trois LED sont éteintes.
Les trois LED s'allument 1 seconde.
Après une brève extinction de toutes les LED (1 dixième de seconde), les deux premières restent éteintes et la troisième s'allume une seconde.
Extinction brève, puis LED n°1 et LED n°3 éteintes, LED n°2 allumée une seconde.
Extinction brève, LED n°1 allumée et LED n°2 et LED n°3 éteintes une seconde.
On retourne au début, mais le programme recommence avec un temps d'allumage de 0,8s (8 dixièmes), puis 6 dixièmes, puis 4, puis 2.
Le programme recommence au début.

Projet 2 : Une GuirLED

Le programme que je vous propose de réaliser utilise 5 LED. Je vous laisse choisir les couleurs que vous préférez. L'objectif est de réussir à créer des illuminations variées un peu comme une guirlande.
On essaiera dans les montages de positionner les 5 LED en une ligne assez rapprochée (contrainte de montage). Puis nos LED devront effectuer des "figures" lumineuses variées. Je vais vous proposer 4 figures pour le moment, vous inventerez celles qui vous conviennent !
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.

Le bouton poussoir

Le principe de ce bouton est que lorsque l'on appuie, le courant passe, et lorsque l'on relâche et bien... le courant ne passe plus !

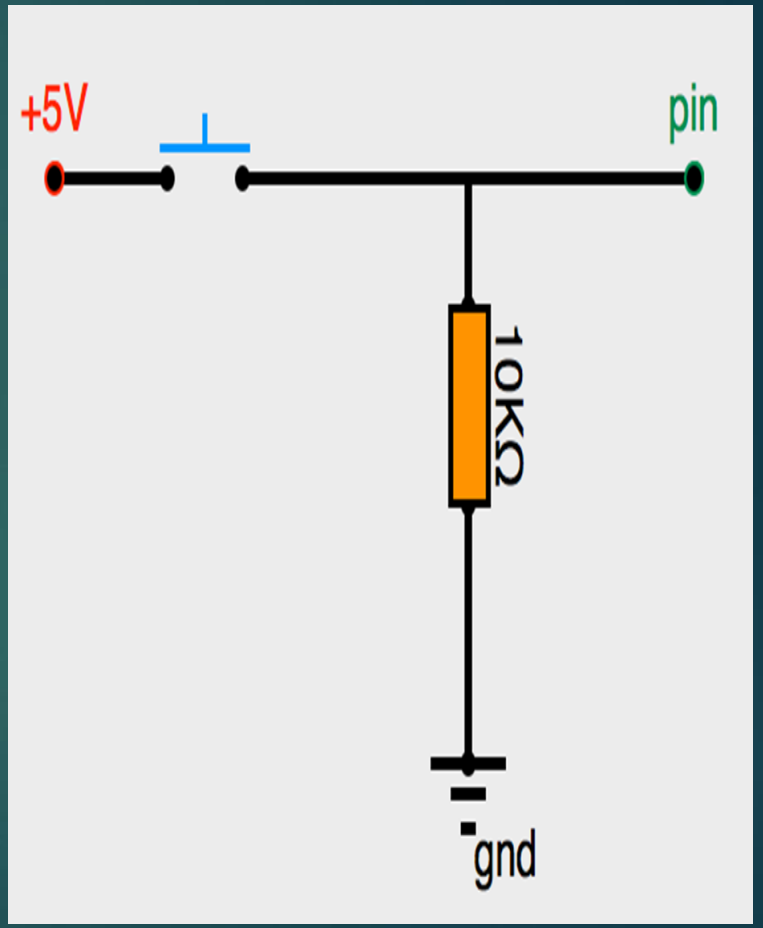
Comment utiliser un bouton par programmation ?

Tout simplement en le reliant à un pin qui est en mode lecture. Si le montage est correctement réalisé, en appuyant sur le bouton, l'Arduino va recevoir l'information et pourra agir en conséquence.
Pour comprendre, l'Arduino va pouvoir lire une valeur de +5V ou de 0V. Donc en théorie, si on envoie le +5V sur un poussoir, quand il est baissé, il laisse passer le courant et l'Arduino reçoit +5V, il indique donc HIGH (ou 1). Si le poussoir est ouvert, l'Arduino devrait ne rien recevoir, donc être à 0V et indiquer LOW (ou 0).

Résistance pull-down

Il faut savoir que l'électricité est paresseuse. Elle va toujours choisir le chemin qui lui résiste le moins. Mais si elle n'a pas le choix, elle passe tout de même là où ça résiste. Nous allons donc ajouter une résistance à notre circuit. Une assez forte pour que le courant ne passe que s'il y est obligé (souvent de l'ordre de 10kΩ).

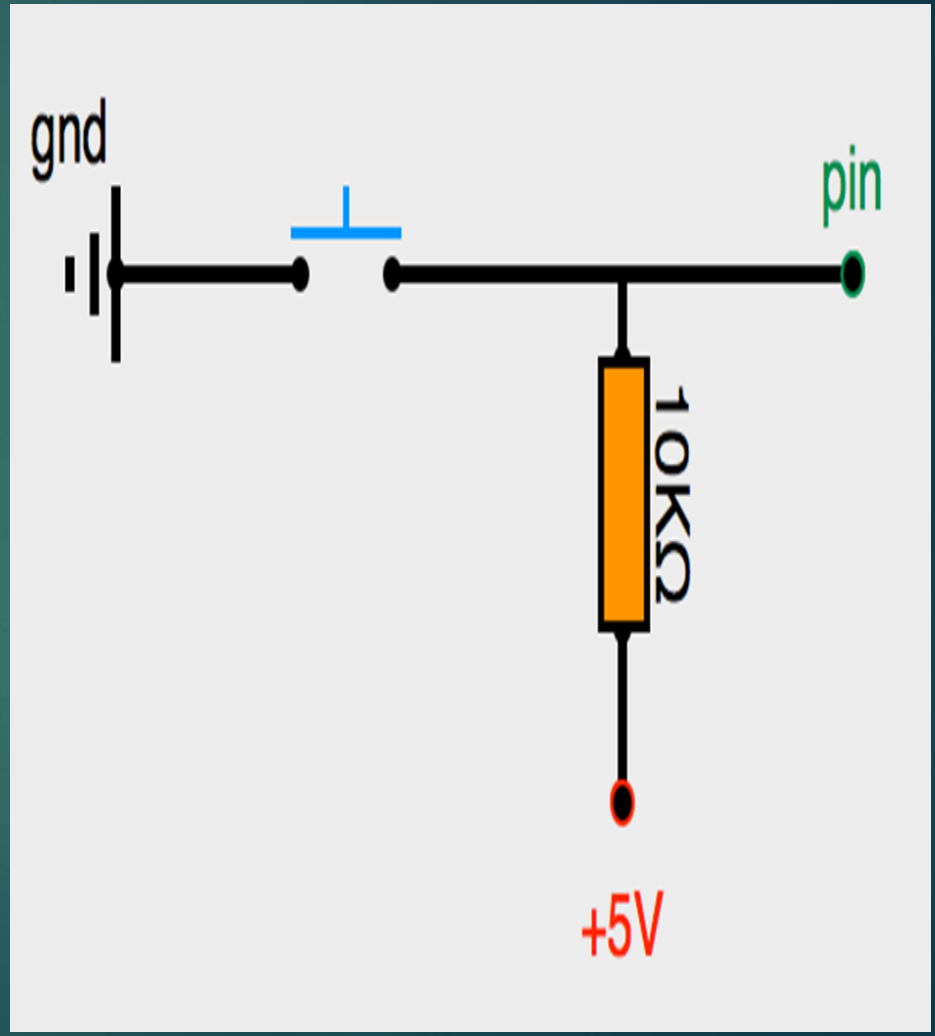
Si le poussoir est baissé, le courant va du +5V au pin de l'Arduino. Il ne prendra pas le chemin du ground car la résistance lui demande un effort. Le pin de l'Arduino recevra du +5V et indiquera HIGH (ou 1).
Si le poussoir est levé, donc le circuit ouvert, le très faible courant résiduel qui sortira du pin de l'Arduino sera absorbé par le Gnd, le pin sera donc bien en LOW (ou 0).



Résistance Pull-Up

Il est possible dans d'autres cas de monter la résistance, non pas vers le ground, mais vers le +5V. Il faut penser à connecter le poussoir au ground (et non plus au +5V) pour que tout fonctionne.

Le fonctionnement en pull-up donne en lecture l'opposé du fonctionnement en pull-Down. C'est important à savoir car dans un cas (pull-down), on obtient HIGH avec le bouton appuyé et dans l'autre cas (pull-up), on obtient LOW avec le bouton appuyé.



Vous voyez donc que la commande pinMode sert finalement à préparer un pin dans trois modes différents :

-Mode **OUTPUT** : l'Arduino fournira par programmation du +5V ou du 0V.

-Mode **INPUT** : l'Arduino lira l'état sur le pin concerné : HIGH (ou 1) pour +5 V reçus et LOW (ou 0) pour 0 V. Il faut prévoir le montage qui correspond.

-Mode **INPUT_PULLUP** : l'Arduino lira les informations reçues, mais le pin sera connecté en interne (dans la carte elle-même) avec une résistance de 20KΩ en mode pull-up.

Attention, pour le pin 13, il est déconseillé d'utiliser le mode INPUT_PULLUP. Si vous devez vous servir du pin 13 comme pin de lecture, préférez un montage avec une résistance externe en pull-up ou pull-down. L'explication se trouve dans le fait que le pin 13 est aussi lié à une LED et une résistance. Il ne fournira donc pas du +5V, mais du +1.7V à cause de la LED et de la résistance en série qui baissent la tension. De ce fait la lecture sera toujours à LOW.

Et bien nous pouvons attaquer nos nos projets pratiques maintenant...

Infographic Style

Le montage contient un bouton poussoir connecté au pin 2 avec une résistance montée en pull-down ;

Une LED (LED1) connectée au pin 4 (pensez à la résistance) ;

Une LED (LED2) connectée au pin 6 (pensez à acheter du pain...) ;

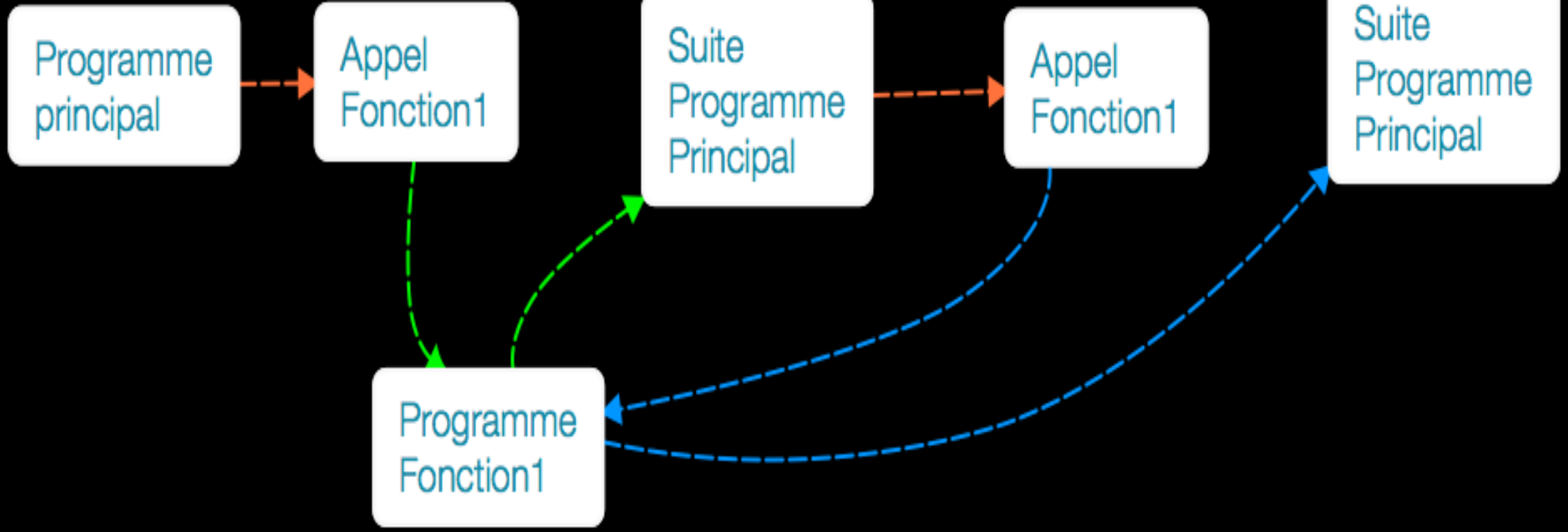
Lorsque le bouton est levé, la LED1 est allumée, la LED2 est éteinte ;

Programme "Jour/Nuit"

Lorsque le bouton est appuyé, la LED1 est éteinte, la LED2 est allumée

Indice : il faudra tester l'état du bouton poussoir (pin en lecture) et faire agir le programme en fonction.

Les fonctions



Declaration de la fonction

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

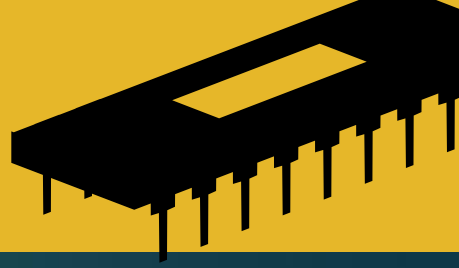
type : c'est le type de variable que la fonction peut renvoyer. Pour le moment on indiquera “void” qui veut dire “vide”, c’est-à-dire que la fonction comprend l’exécution d’un certain nombre d’instructions mais ne renvoie aucune valeur de sortie.

nomDeLaFonction : c'est le nom que l'on choisit pour la fonction. Dans l'exemple d'avant j'ai choisi "fonction1" et "fonction2", mais j'aurais pu mettre "patate" ou "affichageDuNombre".

() : ces deux parenthèse sont obligatoires. Nous allons voir plus loin à quoi elles servent. Si vous les oubliez, l'IDE vous retournera une erreur.

{} : le code du programme de la fonction doit être mis entre accolades. Vous noterez qu'il n'y a pas de point-virgule après l'accolade de fermeture.

Appel de la fonction



Pour appeler une fonction ensuite, il suffit de mettre ce code au bon endroit dans votre programme :
nomDeLaFonction();

Passer une information à la fonction

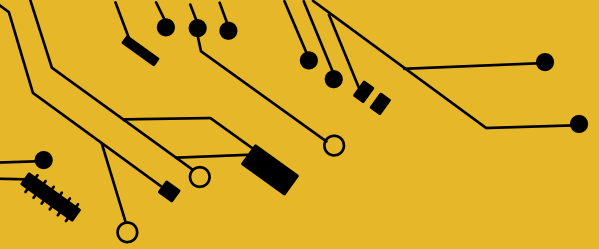
Il faut savoir qu'une fonction peut recevoir une ou plusieurs informations. C'est-à-dire une ou plusieurs variables ou constantes qu'on appelle des paramètres. Il faut pour cela modifier la déclaration de la fonction et son appel.

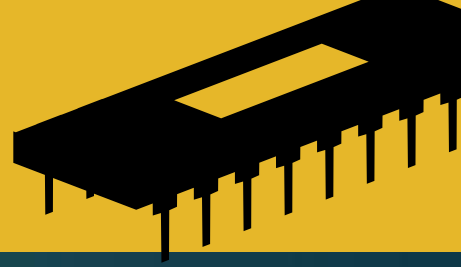
Déclaration :

```
type nomDeLaFonction(type unParametre){
    //code de la fonction qui peut utiliser la variable 'unParametre'
}
```

Appel :

```
nomDeLaFonction(parametre);
```





Il faut que le paramètre envoyé à la fonction soit du même type que ce qui a été déclaré dans la fonction.
Il est possible de passer plusieurs paramètres à une fonction, il suffit simplement de les déclarer à la suite les uns des autres en les séparant d'une virgule :

Déclaration :

```
type nomDeLaFonction(type param1,type param2,type param3...){
    code de la fonction
}
```

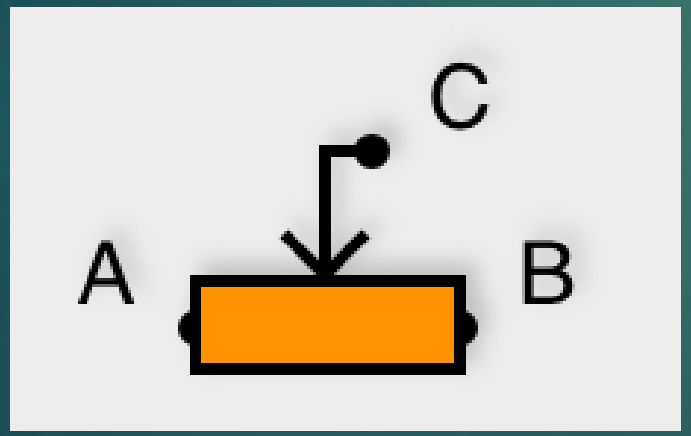
Appel :

```
nomDeLaFonction(param,autreParam, encoreAutreParam,...);
```

Si une fonction peut prendre plusieurs paramètres, elle ne peut en revanche retourner qu'une seule valeur (de type défini avant le nom de la fonction).

Le potentiomètre analogique

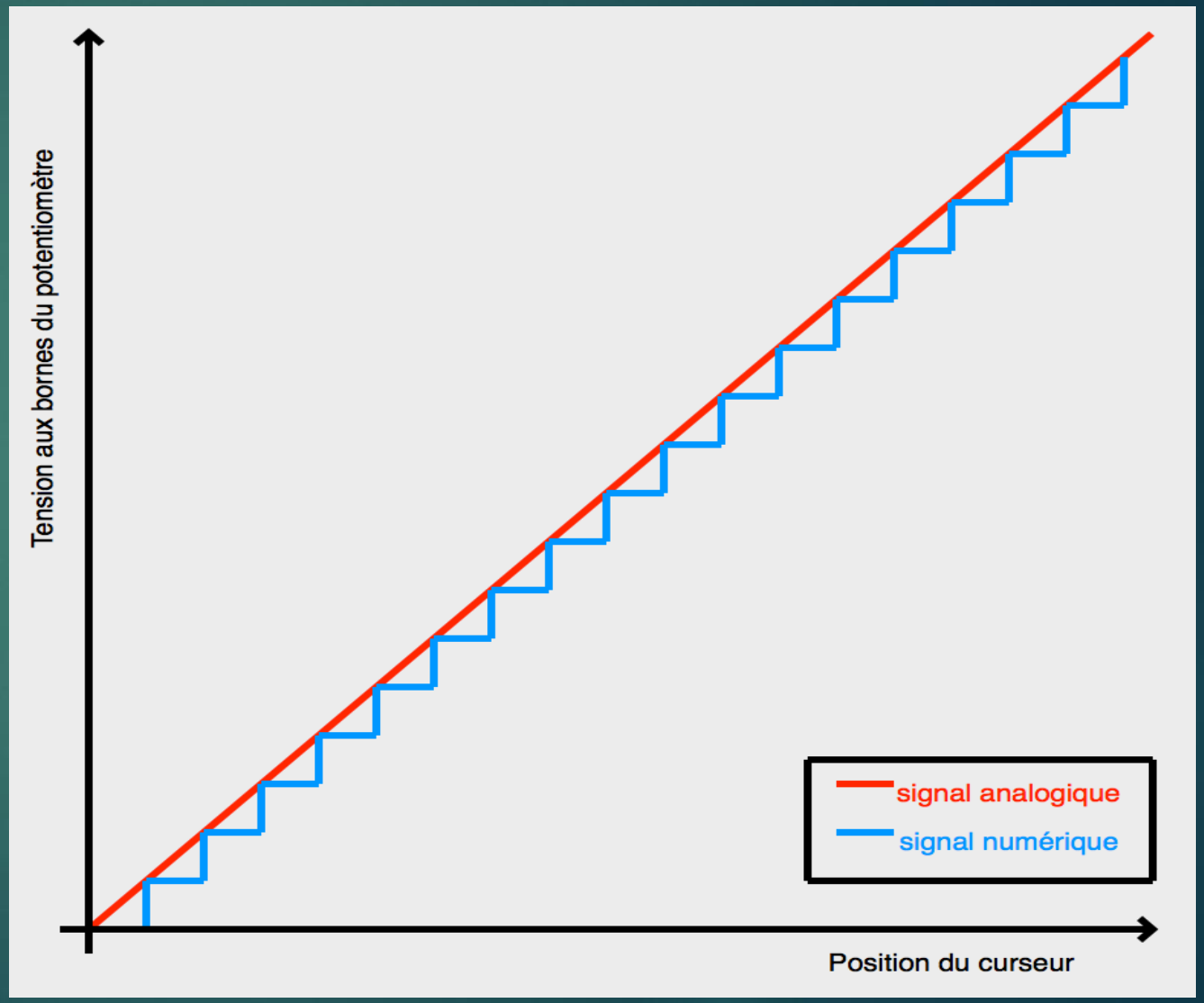
Le principe est assez simple : un curseur se déplace sur une piste résistante (au passage de l'électricité) et permet de faire varier la résistance du potentiomètre. Donc l'électricité entre par un côté du potentiomètre et sort par le curseur. Si le curseur est proche de l'entrée, la résistance est faible et si le curseur est éloigné de l'entrée, la résistance et grande.



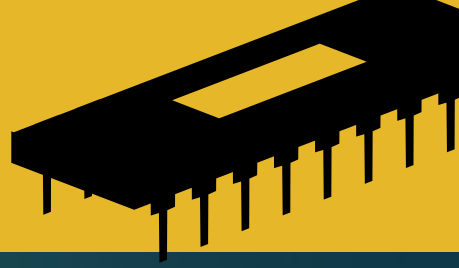
Les entrées analogiques

Pour transformer un signal analogique en signal numérique, il nous faut un **convertisseur analogique numérique**, qu'on appelle plus simplement **CAN**

L'Arduino a tout de même une limite de définition, c'est à dire de résolution du signal. En effet, il ne peut transformer un signal reçu qu'en nombre compris **entre 0 et 1023**, soit 1024 valeurs possibles. Ce qui, vous le verrez, est déjà très bien.
L'autre limite de l'Arduino est que le signal reçu ne doit pas être supérieur à **+5V**.



Le mappage de valeurs



I faut savoir que le langage de l'Arduino propose directement une fonction qui va nous éviter tous ces calculs !

Elle se présente sous cette forme :

map(valeur , min , max, transMin, transMax);
avec :

valeur : c'est la valeur que vous voulez transformer (la tension du potentiomètre du potentiomètre dans notre cas)

min et max : c'est la plage de valeurs (le minimum et le maximum qu'elle peut prendre, soit ici 0 et 1 023 qui correspond à la plage de résolution du CAN de l'Arduino)

transMin et transMax : c'est la plage de valeurs dans laquelle on doit transformer la valeur (ici 1 et 5, car nous attendons 5 positions différentes)

Le joystick

Nous allons profiter d'avoir aborder les potentiomètres pour parler du joystick (oui Lukas, j'arrive aussi à traduire) que tous les gamers ont entre leurs mains.

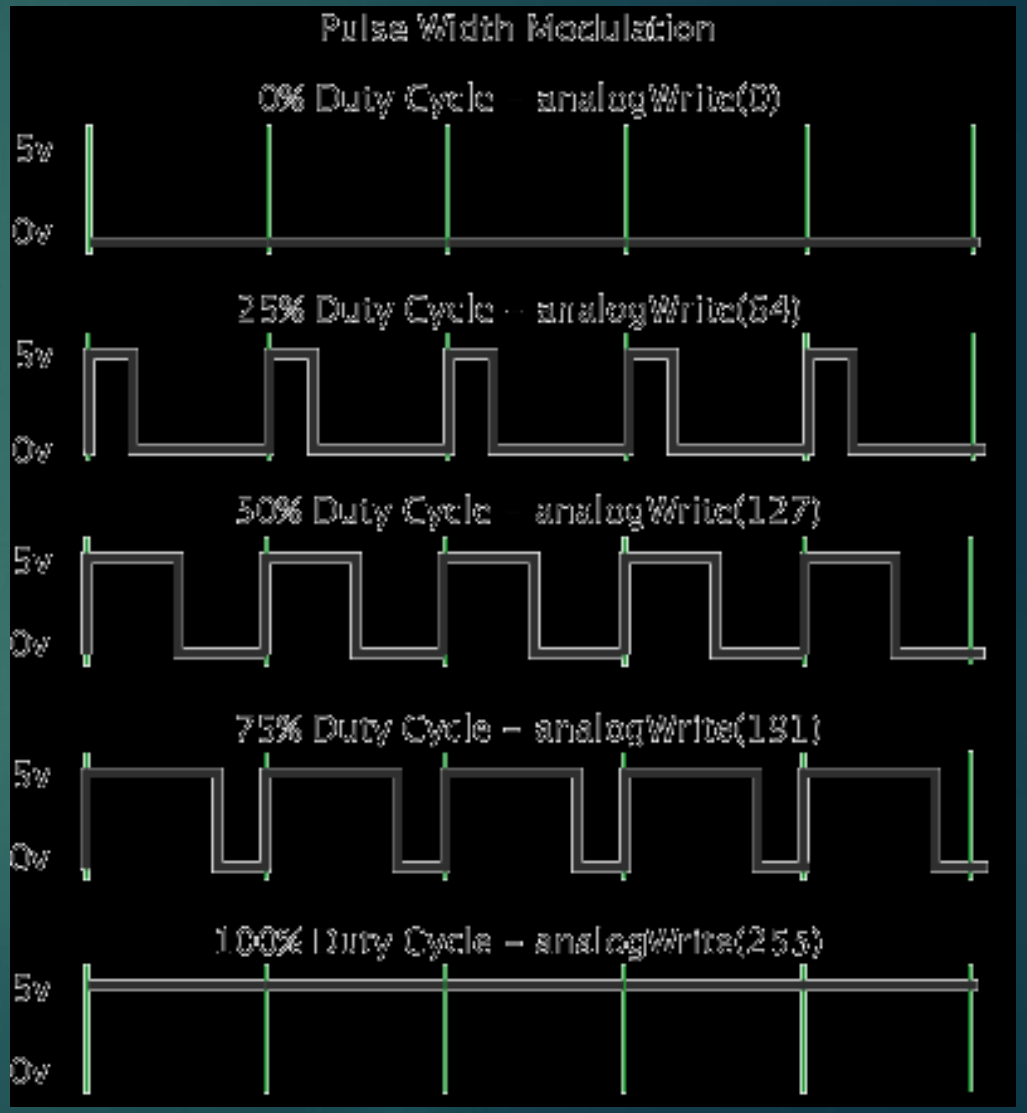
Autrefois, le joystick ne permettait pas une grande précision dans le mouvement. En effet, il s'agissait d'un manche pouvant bouger sur deux axes : avant, arrière, droite et gauche. À chaque mouvement, le manche actionnait un interrupteur qui envoyait un signal binaire à l'ordinateur (quand l'axe était déplacé).

Grâce à l'échantillonnage numérique, les joysticks ont donc gagné en précision. En effet, au lieu d'interrupteurs simples, chaque axe est relié à un potentiomètre (un pour avant/arrière, un pour droite/gauche).

Ce qui permet maintenant, non plus d'avoir un résultat binaire (axe activé ou non) mais une grande nuance de positions possibles entre les axes avant et arrière, et droite et gauche.

MLI (PWM)

Un signal MLI (Modulation de Largeur d'Impulsions) ou PWM en anglais (Pulse Width Modulation) est un signal dont le rapport cyclique varie. Ce type du signal est souvent utilisé dans les applications à valeur moyenne variable (Ex : Commande des moteurs, alimentation réglable,…).

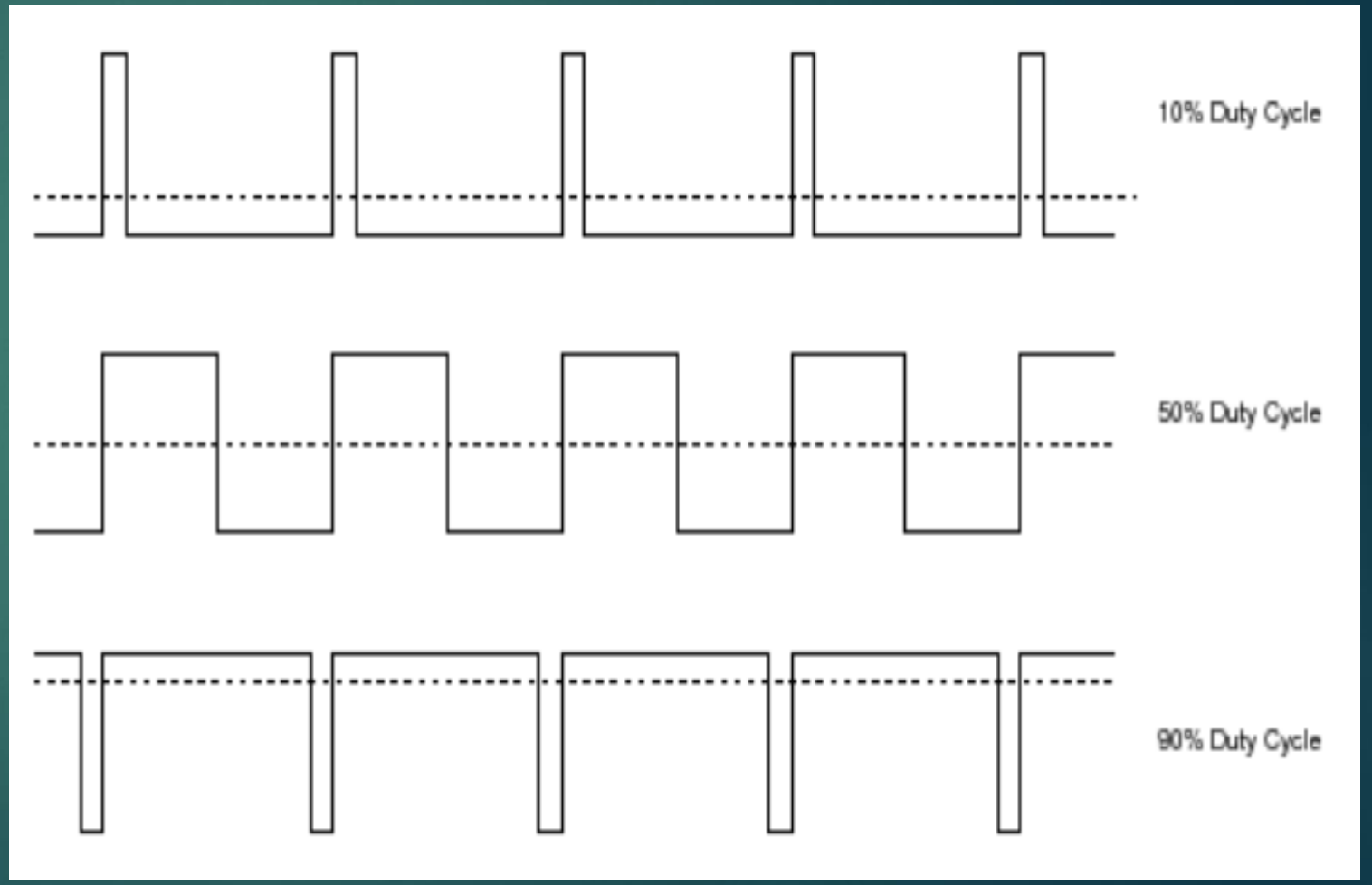


Fonctionnement

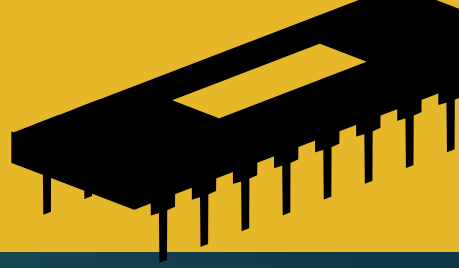
Le signal PWM (MLI, Modulation de largeur d'impulsion) est un signal de fréquence constante et de rapport cyclique variable. Il est mis en œuvre dans des fonctions telles que : La synthèse vocale ou associée à un filtre passe bas permet la synthèse de signaux audio. La commande en vitesse d'un moteur à courant continu, ou ce dernier fait naturellement office de filtre passe bas. La commande en position d'un servomoteur La génération de signaux aléatoires ou périodiques, pour un onduleur par exemple. Courbe PWM dont la valeur moyenne est une courbe sinusoïdale. La valeur moyenne est récupérée simplement par un filtre passe bas.

Applications

Variateurs de vitesse des moteurs
Convertisseurs: AC/DC, DC/AC, DC/DC, AC/AC
Générateur des signaux
Modulateurs
La conversion numérique-analogique
Les amplificateurs de classe D
Contrôle de puissance



RGB LEDs



jusqu'à ce point, nous avons utilisé des LED qui ne sont qu'une seule couleur. Nous pourrions changer la couleur en changeant la LED. Ne serait-ce pas cool si nous pouvions choisir la couleur de notre choix? Qu'en est-il de la sarcelle, du violet, de l'orange ou même plus ???

LED RGB

Présentation de notre nouvel ami, la LED RGB. Une LED RGB est en fait trois petites LED l'une à côté de l'autre. Un rouge, un vert et un bleu. (Par conséquent, pourquoi il est appelé RVB en FR et RGB en EN). Il s'avère que vous pouvez créer n'importe quelle couleur en mélangeant ces trois couleurs claires.
Vous utilisez la même méthode PWM que nous avons discutée plus tôt dans le chapitre pour chaque partie du rouge, du vert et du bleu. Associons-le à trois potentiomètres pour que nous puissions faire varier chacun d'eux et cela devrait avoir plus de sens.

Exercices

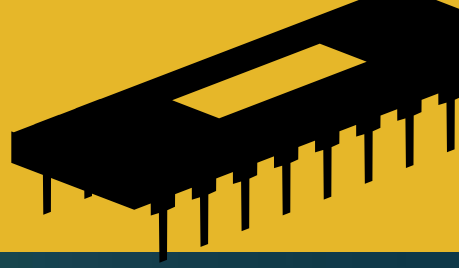
1. Faites des deux boutons poussoirs un bouton «gaz» et «frein». Le bouton «gaz» devrait accélérer le taux de clignotement de la LED, et le bouton «frein» devrait le ralentir.

2. Modifiez la vitesse à laquelle la LED clignote en fonction de la valeur du potentiomètre UNIQUEMENT lorsque vous appuyez sur le premier bouton. (En d'autres termes, vous pouvez régler le potentiomètre, mais cela n'a aucun effet tant que vous n'appuyez pas sur le bouton «ON»

3. CHALLENGE: Utilisez les deux boutons pour enregistrer une couleur «de» et «à». Lorsqu'aucun bouton n'est enfoncé, la LED RVB devrait s'estomper en douceur d'une couleur à l'autre et inversement.

4. CHALLENGE: Pouvez-vous savoir combien de temps il s'écoule entre la dernière instruction de la fonction loop () et la première?

son



Jusqu'à présent, nous avons juste joué avec les lumières. nous ajouterons la création de sons et de musique simples. Afin de produire un son, nous allumons et éteignons le haut-parleur un certain nombre de fois par seconde.
Plus précisément, le la médian (une note de musique) est de 440 Hz. Hz le nombre de fois (ou de cycles) par seconde.)
Donc, tout ce que nous avons à faire pour jouer un A médian est de créer une onde sonore qui tourne 440 fois par seconde. Nous approcherons l'onde sinusoïdale avec une onde carrée (ces termes décrivent simplement la forme). Afin de calculer le temps dont nous avons besoin pour allumer le haut-parleur:
timeDelay = 1 seconde. Cela a un 2 dans le dénominateur car la moitié du temps est avec le haut-parleur allumé et l'autre moitié avec le haut-parleur éteint.
timeDelay = 1 seconde
timeDelay = 1136 microSecondes (une microseconde équivaut à 1 ème de seconde.)

Note simple
Nous avons déjà parlé de la fonction delay (). (N'oubliez pas que les unités sont en millisecondes ou 1 ème de seconde.) Il existe également une fonction delayMicroseconds () (une microseconde équivaut à 1 ème de seconde.) Donc, tout ce que nous devons faire est de configurer notre broche de haut-parleur, puis d'augmenter et de réduire la tension sur cette broche 440 fois par seconde. Rappelez-vous au début de ce chapitre où nous avons compris que nous devons activer (puis désactiver) le haut-parleur pendant 1136 microsecondes. Exécutez ce programme et vous devriez entendre un A (note de musique) qui ne s'arrêtera pas (jusqu'à ce que vous tiriez sur l'alimentation.)

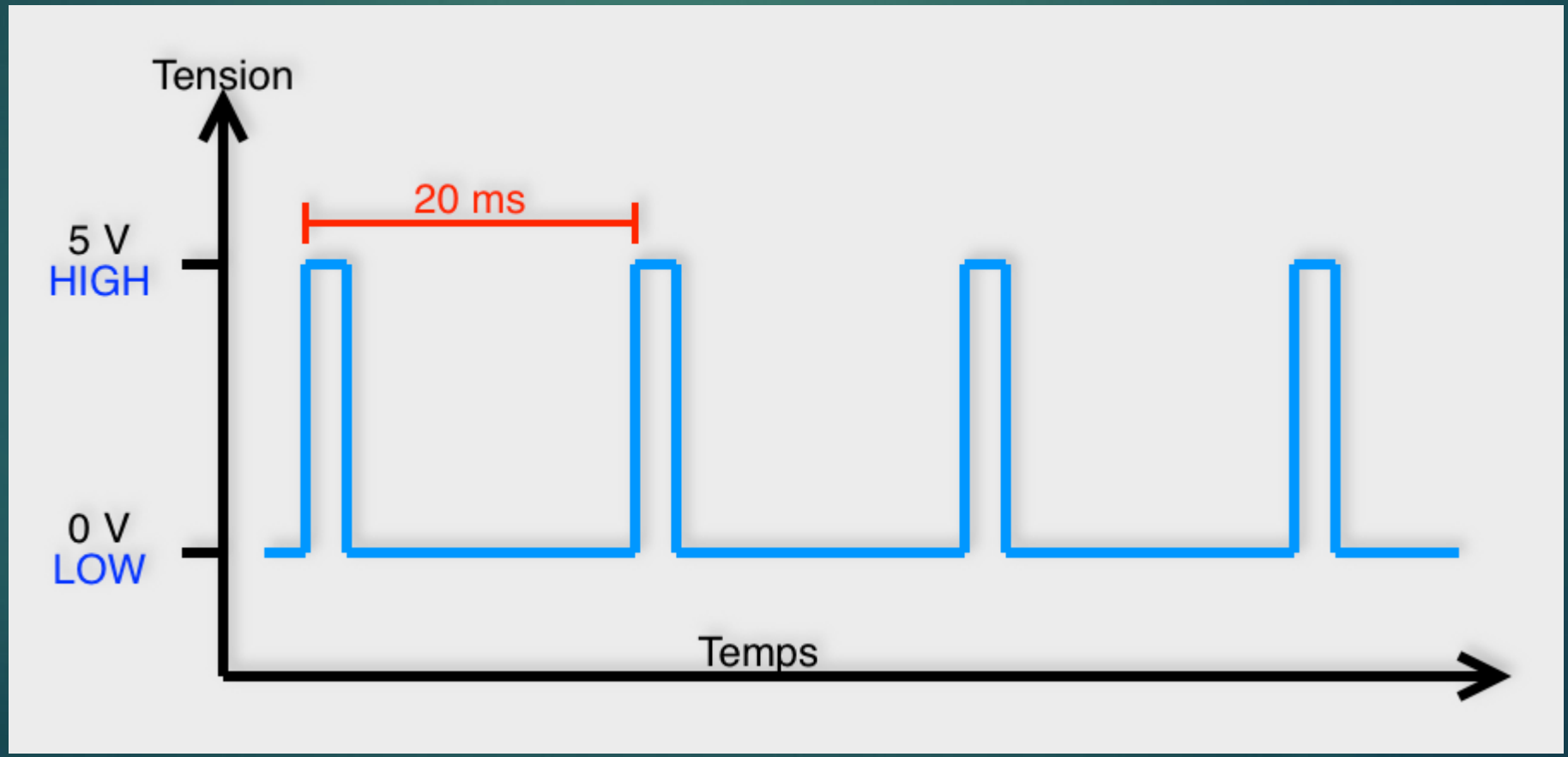
```
const int kPinSpeaker = 9;

const int k_timeDelay = 1136;

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

void loop()
{
digitalWrite(kPinSpeaker, HIGH);
delayMicroseconds(k_timeDelay);
digitalWrite(kPinSpeaker, LOW);
delayMicroseconds(k_timeDelay);
```

Le servo-moteur

Le moteur du servo-moteur en revanche, n'est pas un simple moteur, il est associé à une série d'engrenages qui va lui permettre de gagner un puissance. Mais comme rien n'est gratuit, ce gain en puissance réduit sa vitesse de rotation.

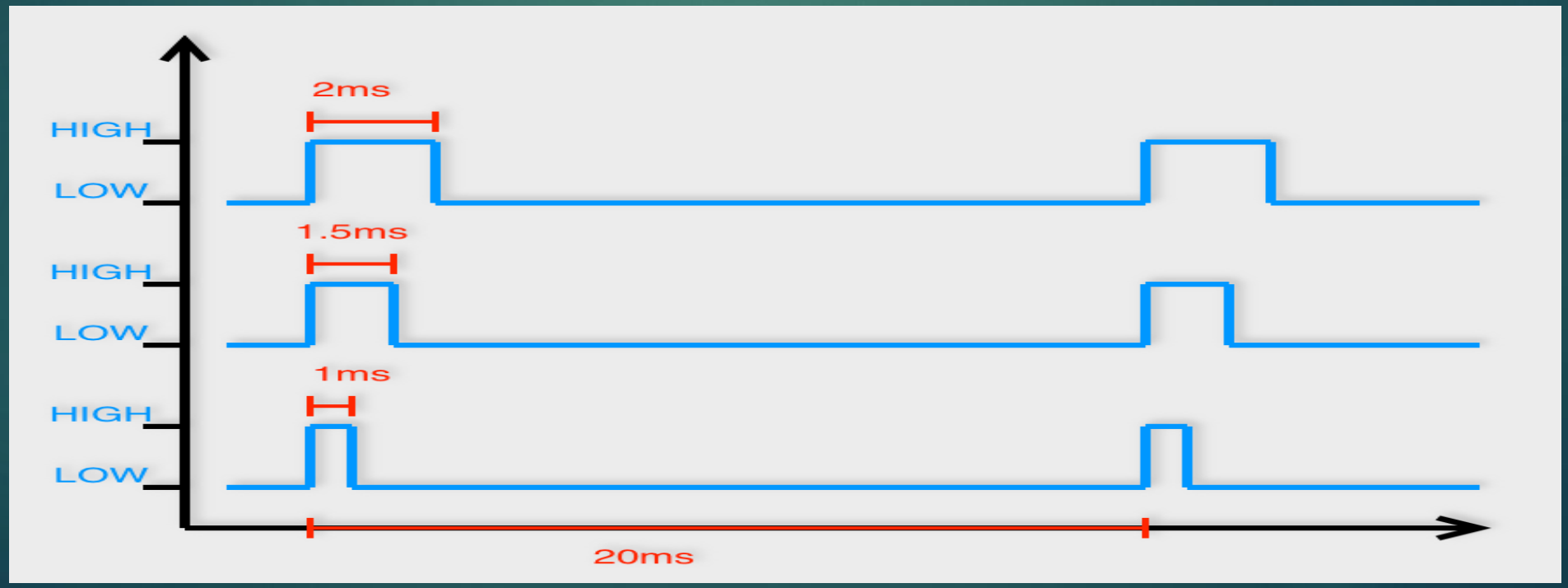
Envoyez des ordres à un servo-moteur

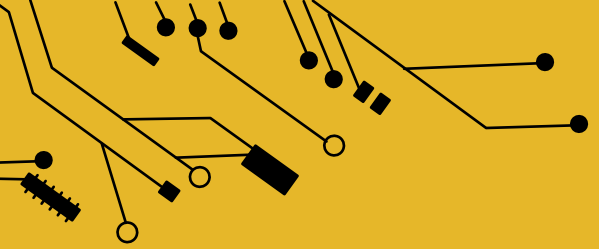
Pour commander un servo-moteur, il faut lui envoyer ce qu'on appelle un train d'impulsions électriques , on peut traduire par : des envois de courant électrique qui se suivent à intervalle et durée précis. L'intervalle de temps entre chaque impulsion envoyée est aussi appelé **période.**

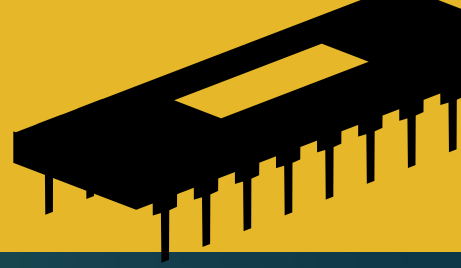


Il faut observer plusieurs choses :

La valeur de la tension est soit +5V (HIGH de l'Arduino) soit 0V (LOW de l'Arduino),
Les impulsions sont envoyées régulièrement, ici toutes les 20 millisecondes. C'est la durée généralement utilisée pour piloter un servo-moteur.
La durée de l'impulsion peut varier. C'est d'ailleurs grâce à elle que nous allons piloter notre servo-moteur.







Et voici comment le servo-moteur va interpréter ces impulsions :

Lors d'une impulsion à 1ms : le servo-moteur se met à la position 0°,

Lors d'une impulsion à 1.5 ms : le servo-moteur se met à la position 90°,

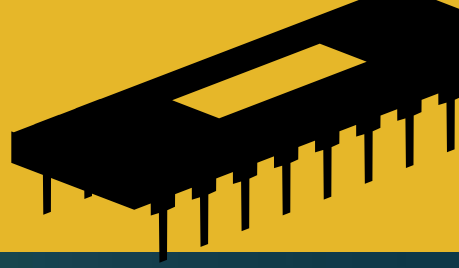
Lors d'une impulsion à 2ms : le servo-moteur se met à la position 180°.

On peut donc facilement faire un mappage des valeurs pour obtenir pour obtenir la durée de l'impulsion en fonction de l'angle :

```
int dureeImpulsion = map (angle,0,179,1000,2000);
```

La valeur de l'angle est entre 0° et 179°, le résultat sera entre 1 000 et 2 000 microsecondes (donc entre 1 et 2 millisecondes). Grâce à cette formule, on peut obtenir toutes les valeurs d'impulsions pour des positions entre 0° et 179°.

#include <servo.h>

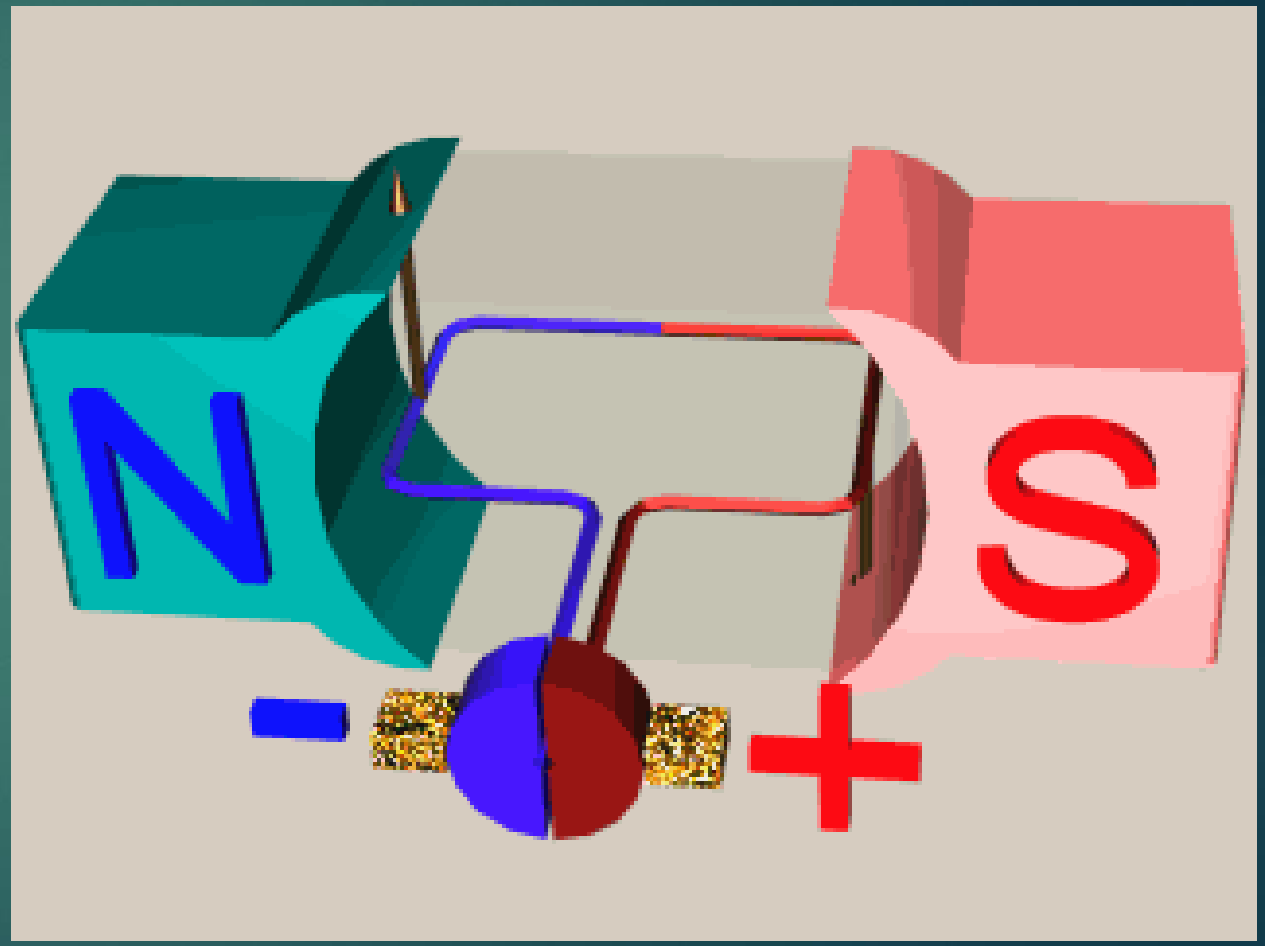


Methods

attach()
write()
writeMicroseconds()
read()
attached()
detach()

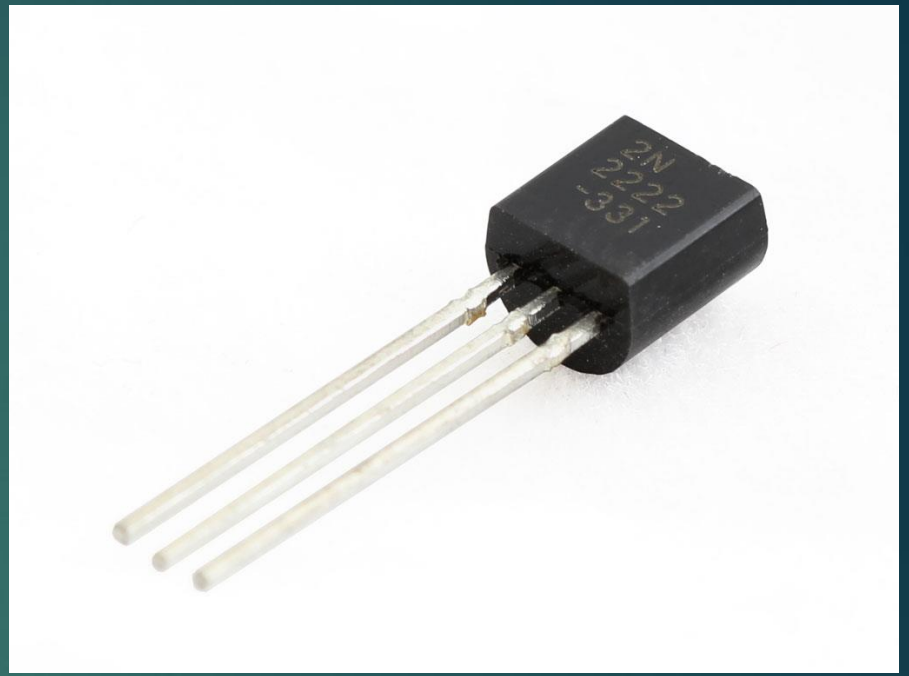
Le moteur à courant continu

le corps du moteur et les aimants statiques qui y sont fixés : le tout s'appelle le **stator** car il ne bouge pas.
Les trois bobines et leur noyau de fer (orange et vert) qu'on appelle le **rotor** car c'est la partie qui tourne.
L'axe qui est la partie qui sera ensuite utilisée pour récupérer ce mouvement de rotation (en entraînant des engrenages, une poulie...).



Le transistor bipolaire

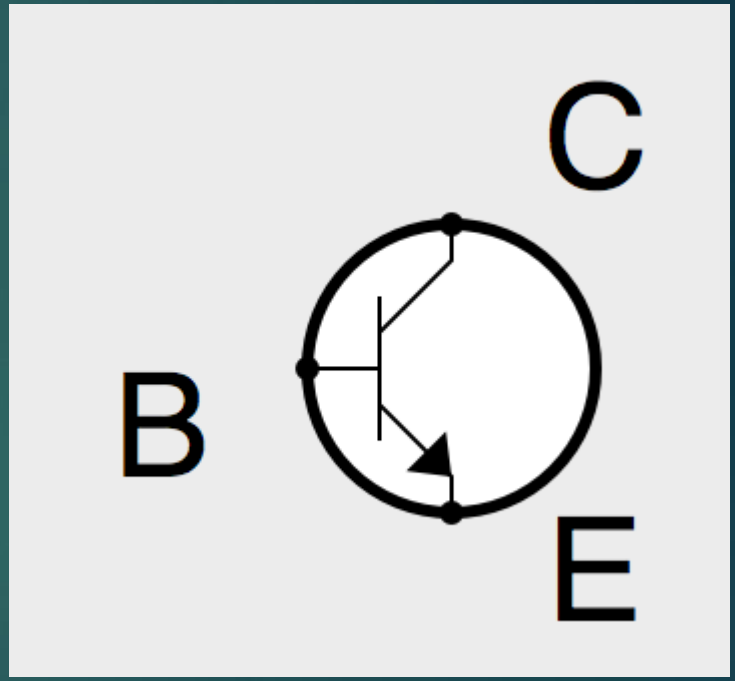
Un transistor bipolaire est un composant électronique. C'est un semi-conducteur (il ne laisse pas passer l'électricité dans n'importe quel sens) qui a été inventé en 1947 et ne ressemblait à l'époque pas du tout à son apparence actuelle. Voici justement un transistor bipolaire de notre époque :



c'est plutôt petit (quelques centimètres de long). Il en existe de plusieurs tailles, dont des minuscules (visibles uniquement au microscope !). Ce composant est à la base de beaucoup de circuits électroniques. Il est utilisé pour deux raisons principales :
Comme amplificateur de signal électrique (ce qui ne nous concerne pas),
Comme interrupteur dans un circuit (ce qui nous intéresse !).

Il existe deux principales familles de transistor bipolaires : les transistors NPN et les transistors PNP.
Je n'entre pas dans les détails de fonctionnement à l'intérieur de ce composant, mais il faut savoir que les transistors NPN sont bien plus couramment utilisés que les PNP.

Le B représente la base : c'est ici que l'on va commander le transistor.
Le C représente le collecteur : C'est cette borne que l'on va relier au moteur.
Le E représente l'émetteur : Cette borne va être reliée au ground.
Pour la connexion, il faut se référer absolument à la datasheet pour repérer les pattes.

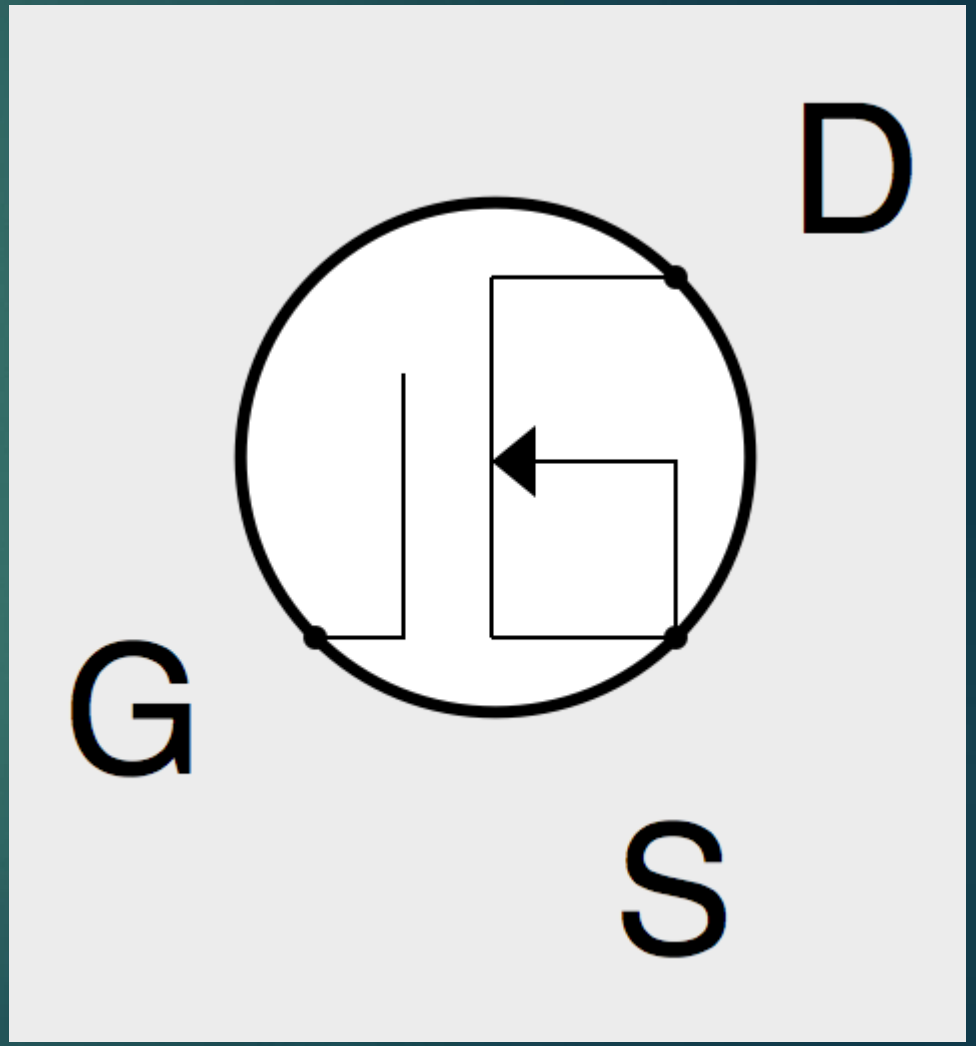


Le transistor à effet de champs ou MOSFET

En fait, le MOSFET est bien plus indiqué pour jouer le rôle d'interrupteur que le transistor bipolaire. Si l'on veut grossièrement expliquer la différence entre l'un et l'autre, le transistor bipolaire est un amplificateur de tension et le MOSFET un amplificateur de courant.

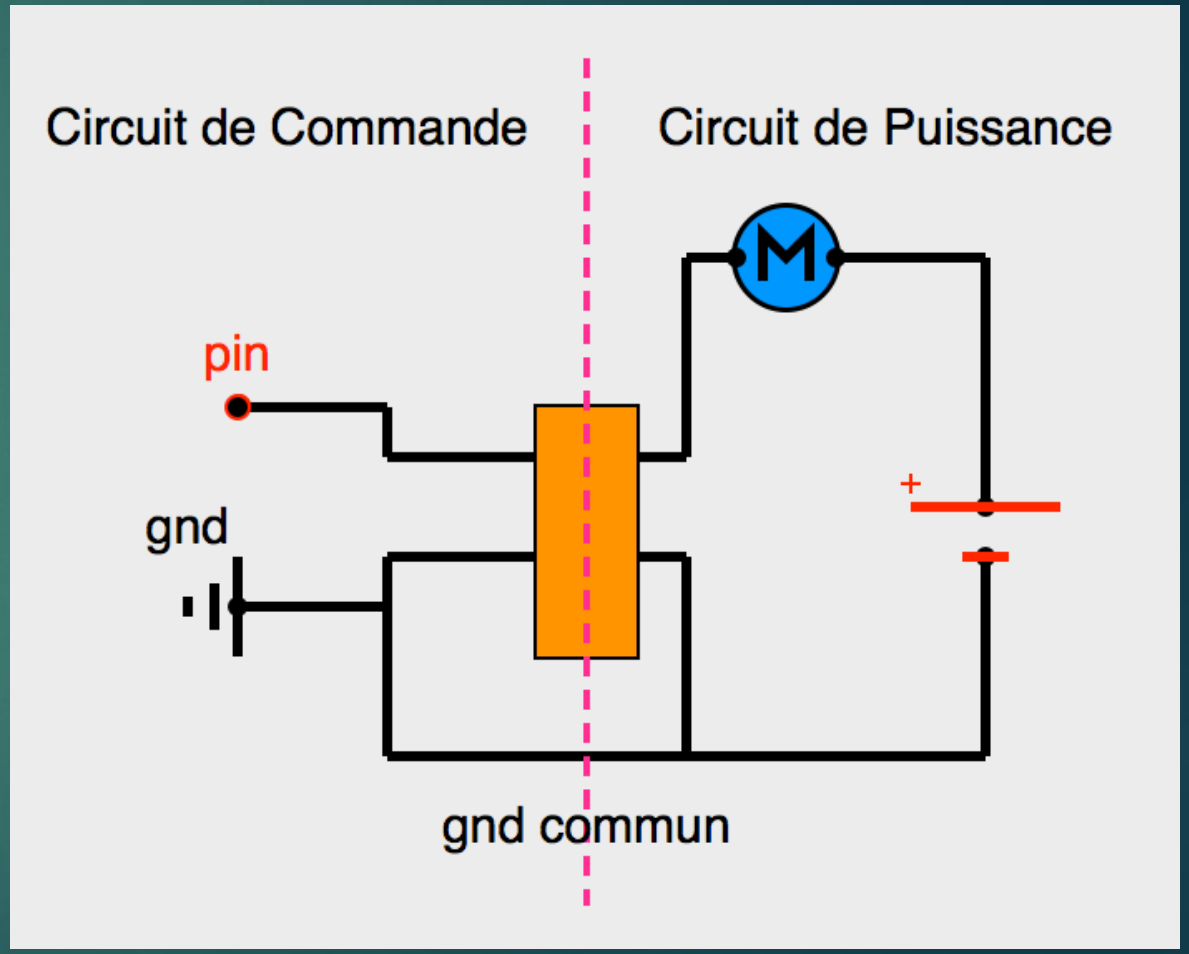
Il suffit donc au MOSFET de recevoir une très faible tension à sa patte de commande pour déclencher le passage du courant. Il est plus stable, il chauffe moins (le métal en haut sert de dissipateur de chaleur), mais il est plus cher (de l'ordre de 10 fois) que le bipolaire. Je vous propose d'utiliser un MOSFET IRF540N.

G représente la borne Gate : c'est ici que l'on commande le transistor.
S représente la borne Source : pour un canal N on la connecte au ground.
D représente la borne Drain : pour un canal N on la connecte au moteur.
Il faut se référer à la datasheet du MOSFET pour le connecter correctement !



Séparez votre circuit de commande de votre circuit de puissance

Avec ce principe de circuit de commande et circuit de puissance, on peut donc piloter des moteurs 12 V, ou 24 V sans risque d'endommagement de la carte Arduino.

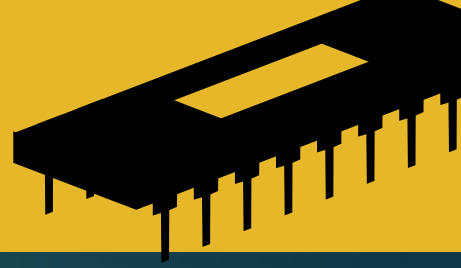


Les capteurs électroniques

Vous avez vu jusqu'à présent que l'on pouvait récupérer des informations depuis l'environnement de l'Arduino grâce à deux commandes : digitalRead(pinDigital) et analogRead(pinAnalogique).

Ce ne sont pas les deux seules commandes mais elles nous permettent de récupérer des valeurs provenant de plusieurs sortes de capteurs qui vont transformer votre Arduino en une machine sensible à son environnement.

Les types de capteurs



Les capteurs tout-ou-rien :

Comme leur nom l'indique, ils ne fournissent que deux données (souvent HIGH et LOW) en fonction de leur état de connexion. On utilise donc pour lire ces données les ports numériques de l'Arduino. On stocke souvent le résultat dans une variable de type booléenne qui sert directement pour les tests.
Les montages utilisés feront souvent intervenir une résistance pull-up ou pull-down pour éviter d'obtenir des valeurs erratiques. L'utilisation de la résistance pull-up interne de l'Arduino est aussi une bonne méthode pour simplifier le circuit, il suffit juste de penser à modifier les tests (voir le chapitre sur le bouton poussoir).

Les capteurs de variation

Ils font varier la tension de sortie soit en résistant, soit en produisant un courant. Ces capteurs seront donc lus avec les ports analogiques de l'Arduino. On transforme alors la valeur de la tension récupérée (souvent entre 0V et 5V) en un nombre entier entre 0 et 1024.
Pour le montage on utilisera souvent une résistance comme pont diviseur de tension. Je n'en expliquerai pas le principe pour le moment, les schémas que je vous proposerai seront suffisants pour correctement placer les composants.

Le capteur de contact

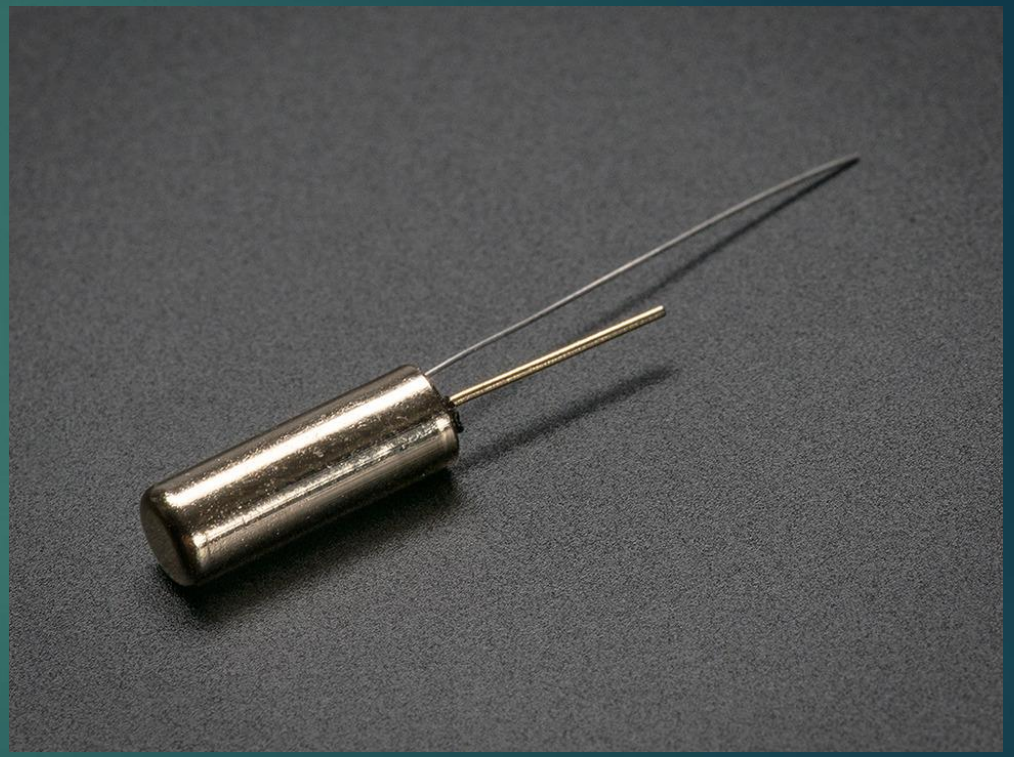
C'est une sorte de bouton poussoir qui s'actionne grâce à la force exercée sur un petit levier. Il en existe de plusieurs formes. En voici quelques uns :

Ce capteur peut avoir 2 ou 3 pattes :
S'il a 2 pattes, c'est un interrupteur simple : soit il est ouvert et il se ferme au contact (type "NO" pour *Normally Open*, c'est-à-dire normalement ouvert) ; soit il est fermé et il s'ouvre au contact (type "NC" pour *Normally Closed,* c'est-à-dire normalement fermé)
S'il a 3 pattes, c'est un inverseur : il laisse passer le courant vers l'une ou l'autre des pattes. Il peut bien sûr être utilisé comme un 2 pattes.

Le tiltsensor

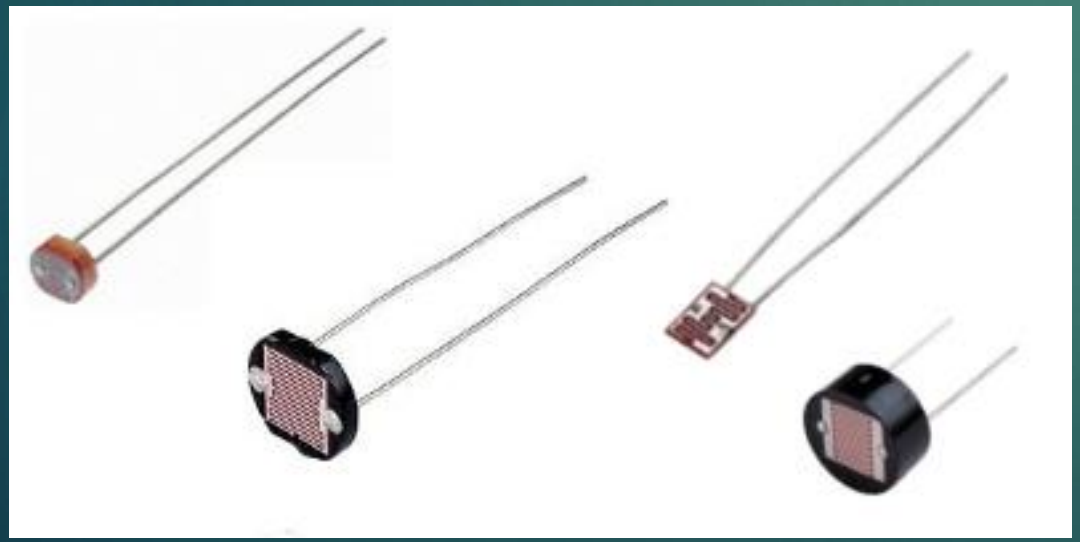
On dit que c'est l'accéléromètre du pauvre. Le principe est simple : une bille en métal vient faire contact sur des pièces métalliques si le mouvement le permet. Ce mouvement doit être brusque (genre le fameux tilt au flipper), ou un changement d'inclinaison. Le tiltsensor fait donc partie des capteurs dits tout-ou-rien.

Grâce au tiltsensor, on peut percevoir un mouvement brusque (chute ou choc) car on capte alors une absence de contact. C'est ce que l'on trouve dans les flippers. On peut aussi lire grossièrement une inclinaison avec deux tiltsensors positionnés en un V très ouvert, l'inclinaison excessive d'un côté ou de l'autre coupera le contact de l'un des deux.



La photorésistance

La photorésistance est une forme de capteur de lumière. C'est un composant très simple à mettre en œuvre et qui permet une interaction intéressante avec l'environnement. Voici à quoi elle ressemble :

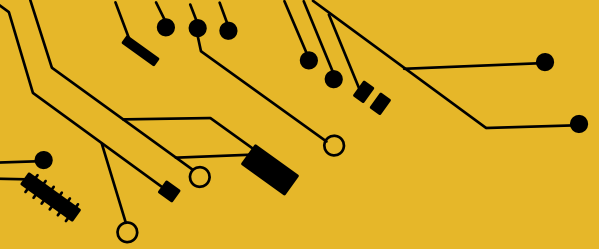


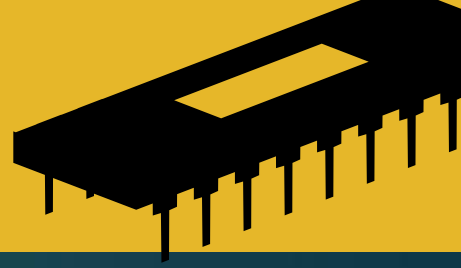
Le principe est assez simple : plus il y a de lumière, plus la résistance est basse. L'obscurité provoque une résistance importante.
Il s'agit donc d'un capteur de variation qu'il faudra connecter à l'Arduino avec un pin analogique. Rappelez-vous le pin analogique transforme une tension reçue entre OV et 5V reçue en valeur entre 0 et 1 024

La photodiode infrarouge

Alors pour faire simple, l'être humain de base (pas les super-héros) ne voit pas tout de la lumière. En fait, la lumière est une onde électromagnétique. Le spectre (c'est son nom) de la lumière visible, donc que notre œil peut voir, va du violet (390 nm) au rouge (780nm). L'unité "nm" veut dire nanomètre, et mesure une longueur d'onde. Le spectre électromagnétique complet comprend dans l'ordre : les rayons gamma, les rayons X, les ultra-violets, la lumière visible, les infrarouges, les micro-ondes, les ondes radio.
La lumière infrarouge est donc invisible pour l'oeil humain. En revanche, nous savons l'émettre et la capter. Elle est utilisée dans vos télécommandes par exemple pour envoyer un message à votre télé

La photodiode infrarouge va réagir à la lumière infrarouge en produisant un signal électrique que l'Arduino va pouvoir récupérer.





Arduino uno

Caractéristiques principales:

Alimentation: - via port USB ou. - 7 à 12 V sur connecteur alim 5,5 x 2,1 mm.
Microprocesseur: ATMega328.
Mémoire flash: 32 kB.
Mémoire SRAM: 2 kB.
Mémoire EEPROM: 1 kB.
Interfaces:
_ 14 broches d'E/S dont 6 PWM.
_ 6 entrées analogiques 10 bits. ...
Intensité par E/S: 40 mA.
Cadencement:16 MHz.

Arduino MEGA

Caractéristiques:

Alimentation:
- via port USB ou
- 7 à 12 V sur connecteur alim

Microprocesseur: ATMega2560
Mémoire flash: 256 kB
Mémoire SRAM: 8 kB
Mémoire EEPROM: 4 kB
54 broches d'E/S dont 14 PWM
16 entrées analogiques 10 bits
Intensité par E/S: 40 mA
Cadencement: 16 MHz
3 ports série
Bus I2C et SPI
Gestion des interruptions Fiche USB B
Version: Rev 3
Dimensions: 107 x 53 x 15 mm

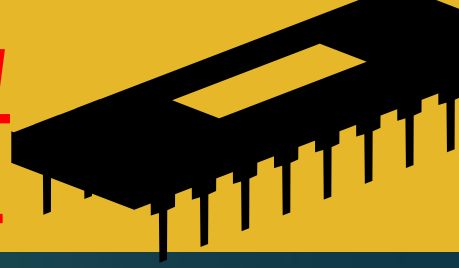
ARDUINO NANO

Caractéristiques:
Alimentation:
- via port USB ou
- 5 Vcc régulée sur broche 27 ou
- 6 à 20 V non régulée sur broche 30

Microprocesseur: ATMega328
Mémoire flash: 32 kB
Mémoire SRAM: 2 kB
Mémoire EEPROM: 1 kB
Interfaces:
- 14 broches d'E/S dont 6 PWM
- 8 entrées analogiques 10 bits
- bus série, I2C et SPI

Intensité par E/S: 40 mA
Cadencement: 16 MHz
Gestion des interruptions
Fiche USB: mini-USB B
Boîtier DIL30
Dimensions: 45 x 18 x 18 mm

Comment sélectionnez-vous la bonne carte pour vos besoins?



Les paramètres à prendre en compte pour vous aider à choisir sont :

Le prix, par ce que ça ne sert à rien de payer plus cher ce qu'on a pas besoin !
La dimension, par ce que si c'est plus petit c'est plus facile à faire rentrer au chausse pied !
Le nombre d'entrée sorties, par ce que si on a besoins de 50 entrées sorties il vaut mieux prendre celle qui en a suffisamment !
Le poids, par ce que si la réduction du poids est la priorité numéro 1 mieux vaut prendre la plus légère !
La taille mémoire, par ce que si on a un programme très lourd mieux vaut qu'il y ai suffisamment de place dans la carte !
Les connecteurs disponibles, par ce qu'avoir les bons connecteurs et des connecteurs supplémentaires c'est toujours plus pratique !
Le besoin de souder les connecteurs, par ce que si tu n'as pas accès à un fer à souder c'est pas pratique !
En fonction de votre projet gardez bien en tête ces paramètres pour bien choisir la bonne carte Arduino qui vous conviendra le mieux ☺