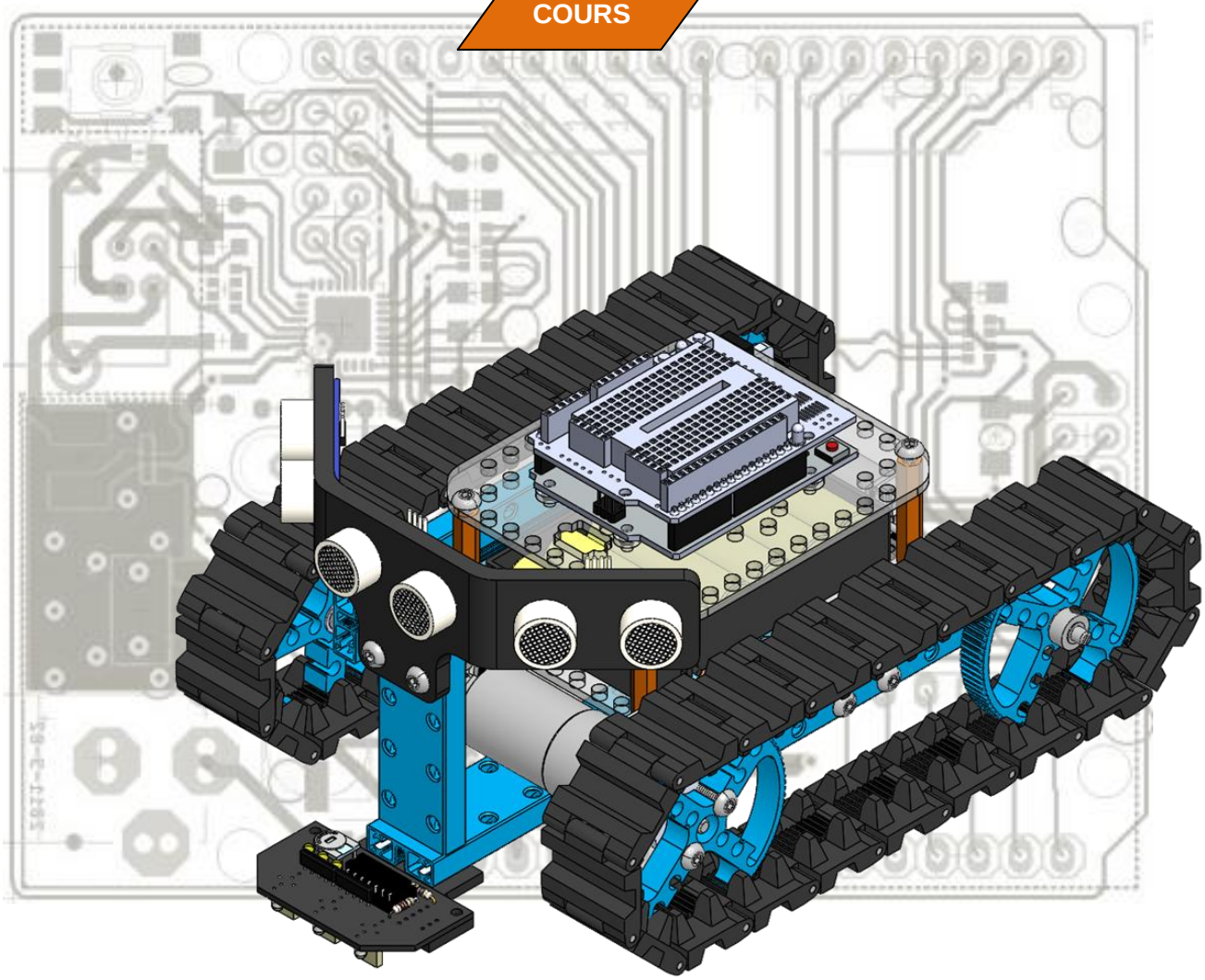
R3.10 - INGENIERIE DES SYSTEMES CYBERPHYSIQUES

COURS



SEMESTRE 3

Génie Mécanique Productique Montluçon

PPN IUT BUT GMP 2ème année

Nom de la ressource	R1.07 - Production - Méthodes		
Semestre	Semestre 3		
Compétence(s) ciblée(s)			
C1-Spécifier Spécifier les exigences technicoéconomiques industrielles	C2-Développer Déterminer la solution conceptuelle	**C3-Réaliser** Concrétiser la solution technique retenue	**C4-Exploiter** Gérer le cycle de vie du produit et du système de production
X		**X**	**X**

Apprentissages critiques

AC21.01 : Traduire les besoins clients en exigences techniques
AC21.02 : Elaborer un document de spécifications pour un process ou un produit industriel en étant guidé
AC21.03 : Réviser les exigences techniques en mode partagé/collaboratif dématérialisé avec le client
AC21.04 : Initier le projet de développement en définissant les principaux jalons
AC22.01 : Situer les éléments d'un système complexe et leurs interactions, dans l'espace, dans le temps.
AC22.02 : Proposer des solutions pertinentes au regard de la taille des séries et de l'aspect économique.
AC22.03 : Combiner des solutions élémentaires avec un encadrement limité.
AC22.04 : Classifier les solutions selon les critères du cahier des charges.
AC24.01 : Mesurer les performances d'un système/produit/ procédé en suivant les procédures (normes, protocoles, recommandations,…)
AC24.02 : Structurer les données existantes associées au système/produit/procédé en suivant les procédures (normes, modèles, standards...)
AC24.03 : Analyser les performances d'un système/produit/procédé en vue de son amélioration

SAÉ concernée(s)	SAÉ3.01
Prérequis	R1.01, R1.04, R1.10, R2.01, R2.10
Descriptif détaillé	Électricité pour les équipements industriels : ● Sécurité : ○ Composants de sécurité de la chaîne de puissance ○ Protection des biens et des personnes ○ Notions de risque électrique ○ Notions d'habilitation électrique ● Actionneurs et récepteurs de puissance : ○ Courant alternatif monophasé et triphasé ○ Puissances en courant alternatif, rendement ○ Principes des moteurs à courant alternatif ○ Technologie et choix des actionneurs, des générateurs électriques ○ Conversion d'énergie ○ Commande et variation de vitesse ● Câblage industriel : ○ Schémas électriques ○ Câblage de démarrage moteur Base de données : ● Structure : ○ Notions d'ERP ○ Structure d'une BDD, Modélisation ○ Import / export de données ● Recherche d'informations : ○ Tri de données, filtrage, requêtes ○ Création de formulaires être évoqués
Mots clés	Equipements industriels, sécurité, câblage, base de données

SOMMAIRE

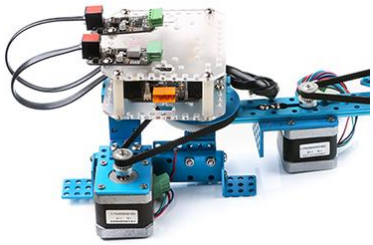
I. Automatisme et microcontrôleurs

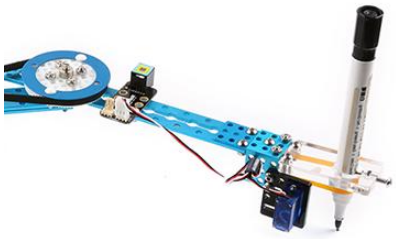
L'automatisme est .....

L'automatisation d'un système peut être réalisée .....

L'Écrivain de Pierre Jaquet-Droz

Assis devant un pupitre, l'automate tient une plume d'oie qu'il trempe dans l'encrier, puis il la secoue légèrement avant de commencer de dessiner les lettres sur le papier. L'Ecrivain est capable de tracer un texte de 40 signes au maximum, répartis sur quatre lignes. La principale invention de son mécanisme est le système de programmation par disque, qui lui permet d'écrire des textes suivis sans intervention extérieure. Il peut écrire n'importe quelle phrase, lettre par lettre.





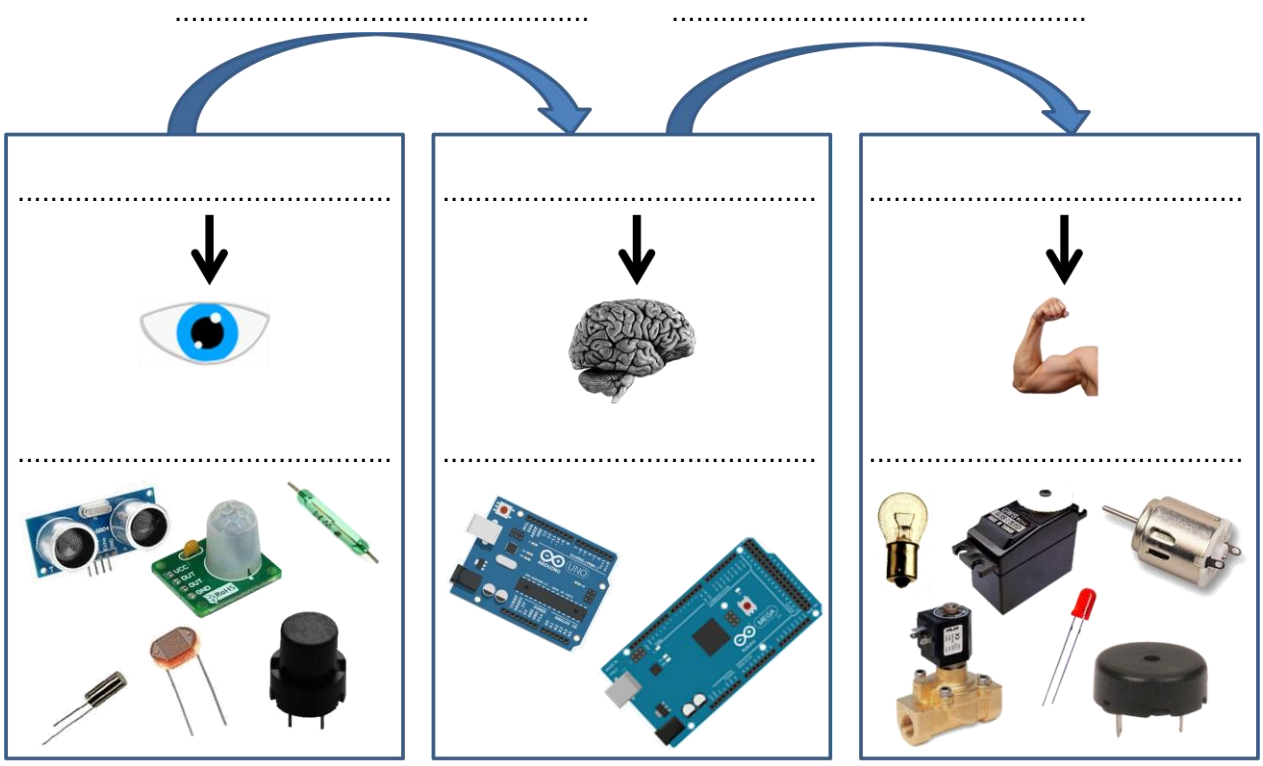
mDrawBot de Makeblock

mDrawBot est un robot articulé à l'aide de moteurs pas à pas. Avec un stylo il peut dessiner des images sur une surface plane. Un logiciel spécialement conçu pour le mDrawBot permet d'importer des images, puis de programmer le **microcontrôleur** pour que le robot les dessine.

II. Chaîne d'information et chaîne d'énergie

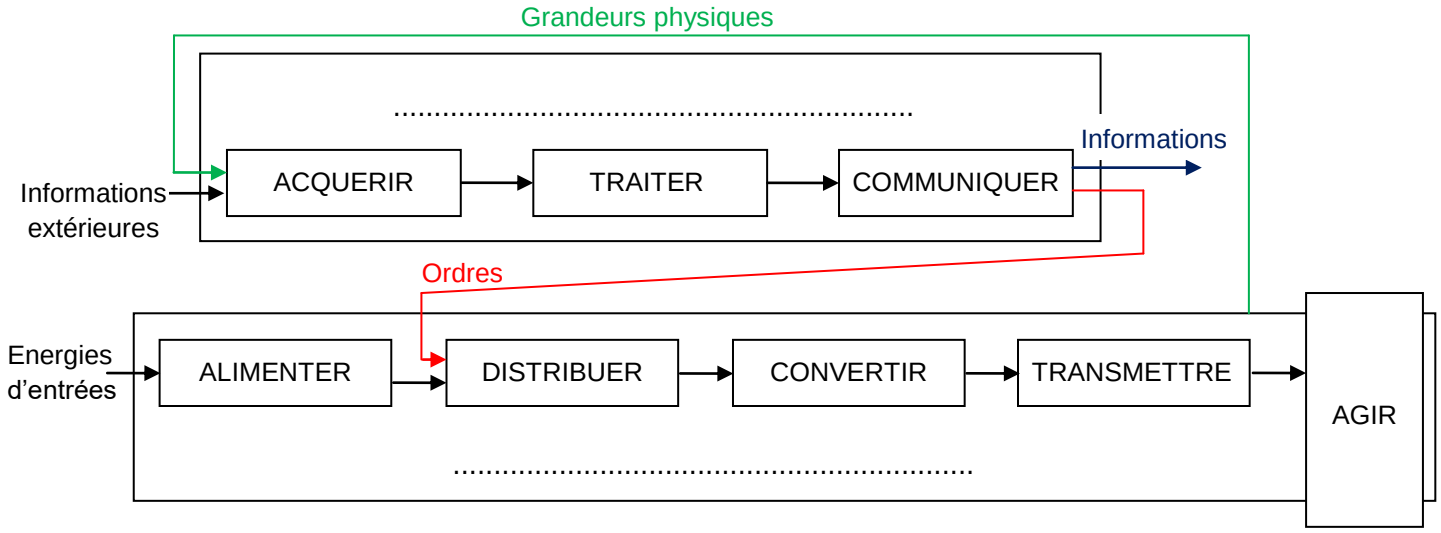
a) Vocabulaire

Un système automatique est composé de plusieurs éléments qui interagissent entre eux, avec un vocabulaire propre à chacun. Pour mieux comprendre leurs rôles et leurs interactions, une analogie avec le corps humain est possible :



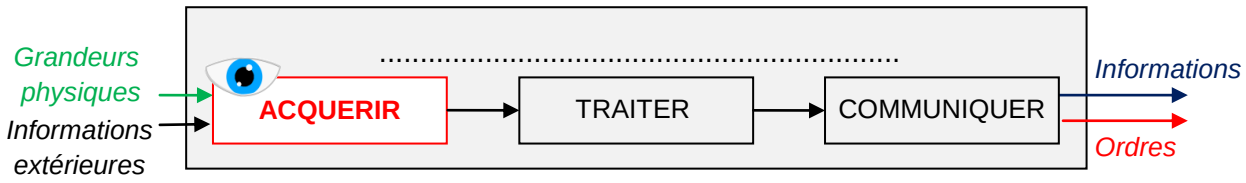
b) Représentation

Les informations et les énergies doivent être transportées et parfois transformées. L'ensemble du système peut donc être représenté par sa chaîne d'information et d'énergie. Cette **représentation** est **normalisée** :



c) Chaîne d'information

1) Acquérir



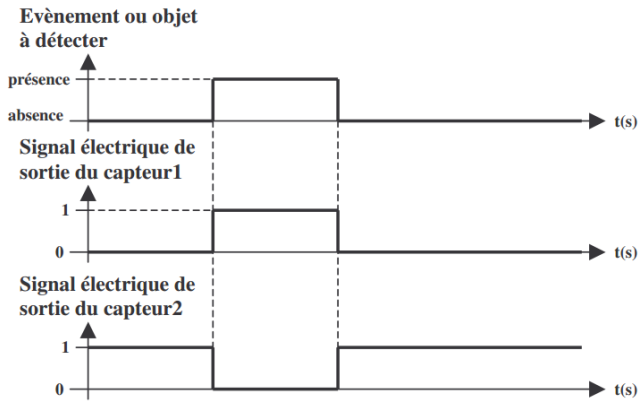
ACQUERIR : .....

.....

Cette fonction est assurée par les capteurs. Les caractéristiques de ce signal (courant, tension, niveaux logiques, valeur moyenne, fréquence, amplitude, nombre binaire,…) dépendront directement de la grandeur physique captée et du capteur utilisé. Il existe 3 grandes familles de capteurs :

- Capteur TOR (Tout Ou Rien)

Le signal électrique en sortie de ce capteur est de type logique (signal acceptant 2 niveaux : niveau logique 0 (niveau logique bas) ou niveau logique 1 (niveau logique haut)).



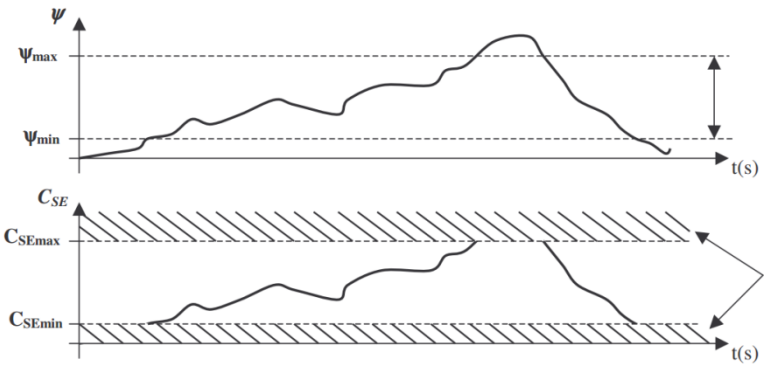
Capteur 1 : TOR délivrant un niveau logique haut lorsqu'il détecte une présence de l'objet

Capteur 2 : TOR délivrant un niveau logique bas lorsqu'il détecte une présence de l'objet

Il faut étudier la **datasheet** (documentation constructeur) du capteur et du câblage de celui-ci, pour trouver dans quel cas le signal de sortie est au niveau logique 1 ou 0.

- Capteur analogique

Une variation de la grandeur physique d'entrée du capteur produit une variation de la caractéristique électrique du capteur (courant, tension, fréquence, valeur moyenne,…).



.....

.....

.....

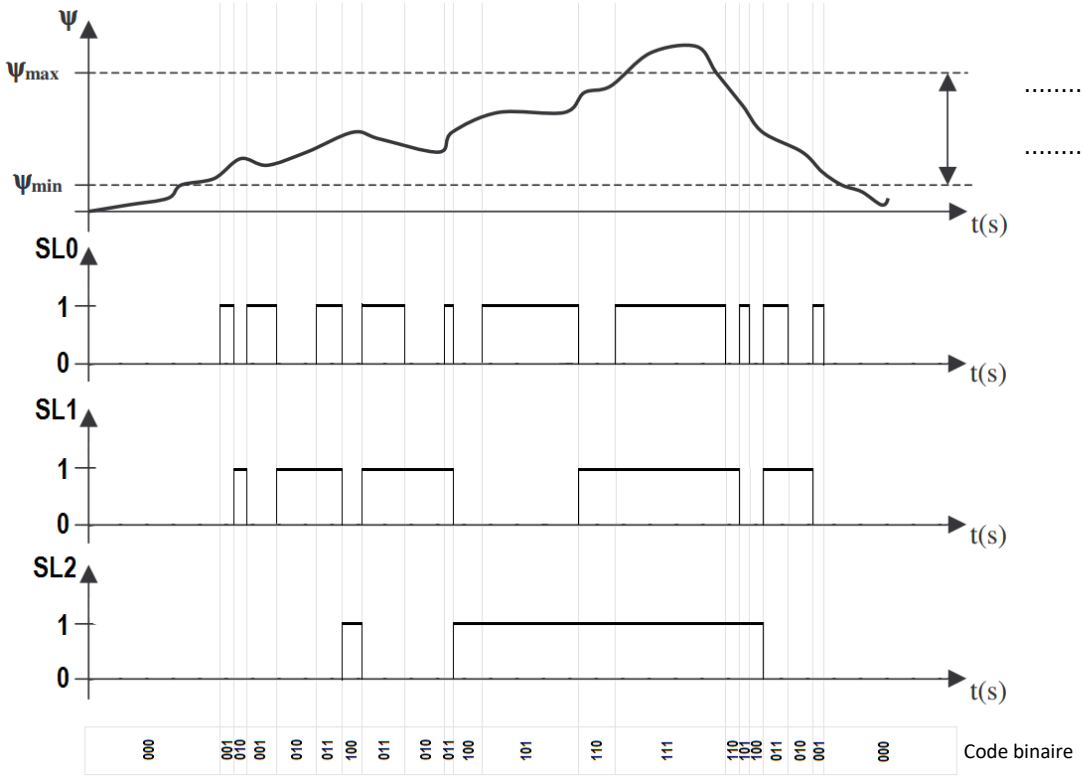
.....

Pour chaque grandeur physique ψ, il existe une valeur du signal électrique C_{SE} (dans l'étendue de mesure du capteur). Pour certains capteurs analogiques, ce n'est pas la caractéristique du signal électrique qui est modifiée mais un composant électronique interne au capteur (résistance, inductance, condensateur,…).

Par exemple, pour une thermistance, la valeur de sa résistance interne varie en fonction de la température de son environnement.

- Capteurs numériques

Ce type de capteur produit un nombre **N** (combinaison de **n** signaux logiques hauts et bas = code binaire) qui dépend directement de la grandeur physique à capter.



.....

.....

Pour une grandeur physique ψ, le capteur va envoyer un nombre binaire de N bits. Dans l'exemple, le capteur code la grandeur physique mesurée sur 3 bits. La valeur dans l'exemple varie donc de « 000 » pour la valeur la plus faible à « 111 » pour la valeur la plus forte.

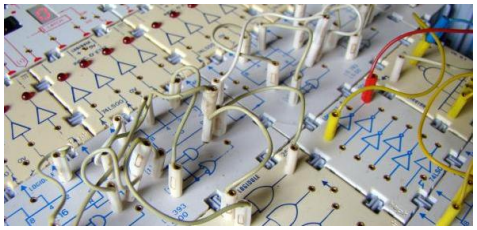
Contrairement à la thermistance vue précédemment, ce capteur de température (DS18B20) va fournir une valeur numérique au microcontrôleur, pour une température comprise entre -55°C et 125°C.

Pour le choix et l'utilisation des capteurs analogiques et numériques, il faut connaitre la nature, la valeur et la précision du signal fourni pour une grandeur physique donnée. Il faut donc réaliser une étude de la datasheet correspondante (détermination de la sensibilité, étendue de mesure du capteur,...).

2) Traiter

Les informations issues de la fonction « acquérir » doivent être **TRAITÉES** pour que des décisions soient prisent. En électronique, il est possible d'utiliser :

- La logique câblée



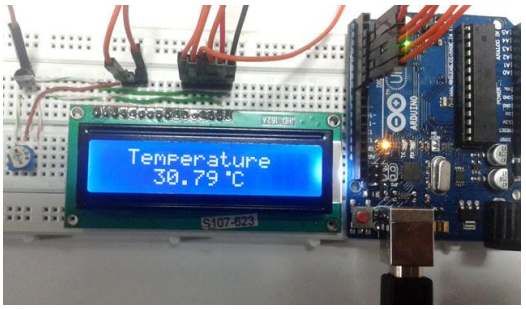
Portes logiques câblées

Un problème est résolu par un ensemble de fonctions logiques (AND, OR, NOT, NAND,…) câblées entre elles. La taille du circuit est croissante avec la complexité du problème. Pour modifier le fonctionnement du système un nouveau circuit doit être créé.

- La logique programmée

Un API (Automate Programmable Industriel) ou un microcontrôleur est programmé en fonction du problème à résoudre. Pour modifier le fonctionnement du système, il suffit de modifier le programme.

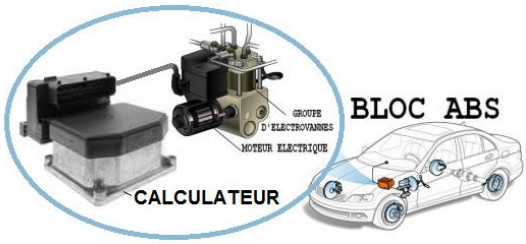
Arduino programmé pour afficher la température



Un **microcontrôleur** (en notation abrégée µc, ou uc ou encore MCU en anglais) est un circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires (mémoire morte pour le programme, mémoire vive pour les données), unités périphériques et interfaces d'entrées-sorties.

Le **microcontrôleur**, moins coûteux qu'un API est présent partout de nos jours, comme par exemple dans :

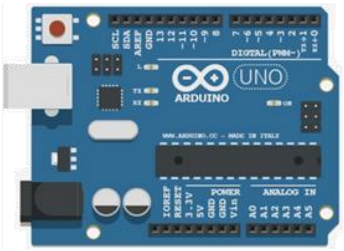
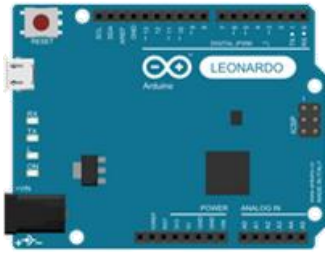
- les maisons : machine à laver, alarme, jouets,…
- les voitures : airbags, injection, ABS,…
- l'industrie : machines outils, bras robotisés,…

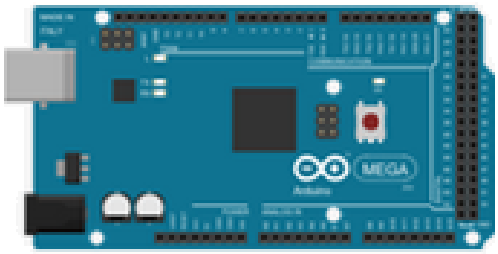
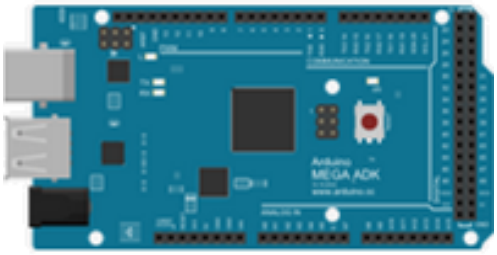
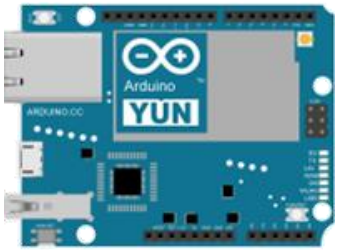
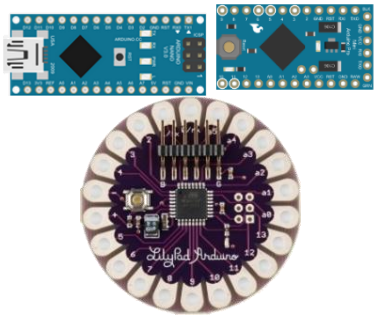


Pour plusieurs raisons, nous allons, cette année, utiliser un microcontrôleur appelé **Arduino** :

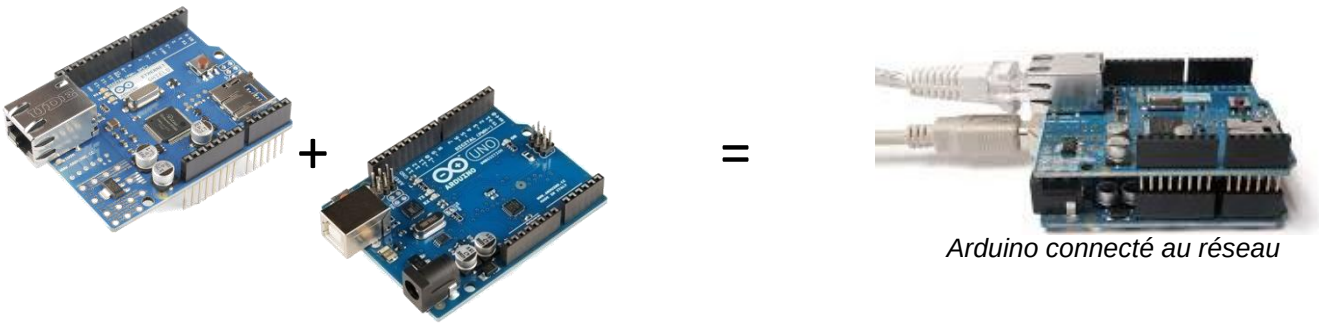
- .....
- .....
- ..... :
  - .....
  - .....
- ..... :
  - .....
  - .....

La famille Arduino est assez grande et comporte plusieurs modèles de cartes différentes : Uno, Méga, Dur, Léornado, Mega ADK, Yun, Lilypad, Nano, ProMini,… Chaque carte a ses avantages et inconvénients. Il faut choisir la plus adaptée à notre besoin :

Nom	Carte	Avantages	Inconvénients
Uno			
Léonardo			

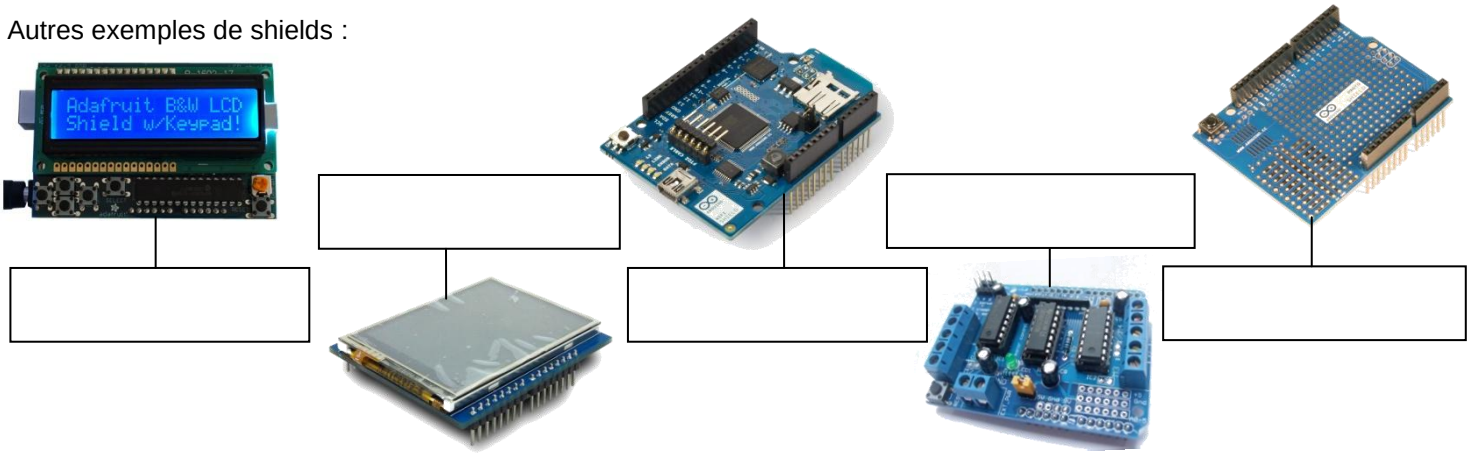
Nom	Carte	Avantages	Inconvénients
Mega			
Mega ADK			
Due			
Yun			
Nano **Pro Mini** **Lilypad**			

Pour augmenter les fonctionnalités d'une carte Arduino, il existe les« **shields** ». C'est une carte qui se connecte directement sur l'Arduino sans câblage. Exemple avec le Shield Ethernet et l'Arduino Uno :



Arduino connecté au réseau

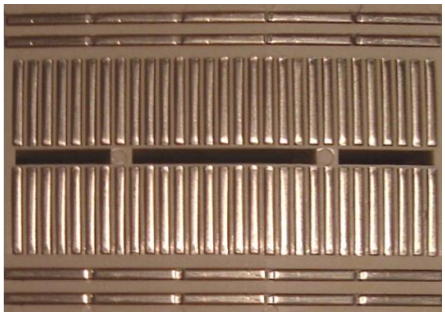
Autres exemples de shields :



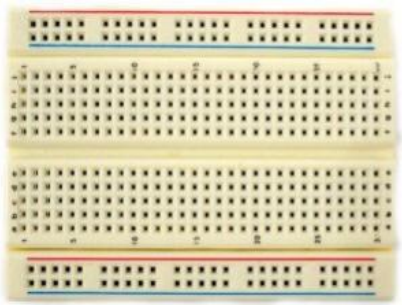
Avant de souder des composants sur un shield de prototypage, le montage sera testé sur une **breadboard.**

BREADBOARD : .....

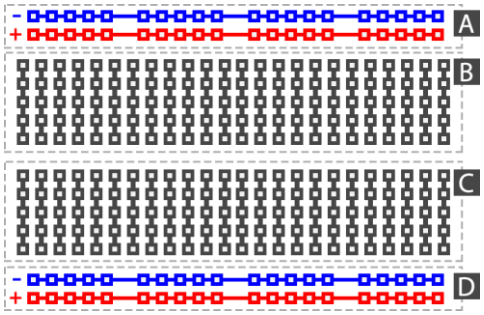
.....



.....



.....



.....

.....

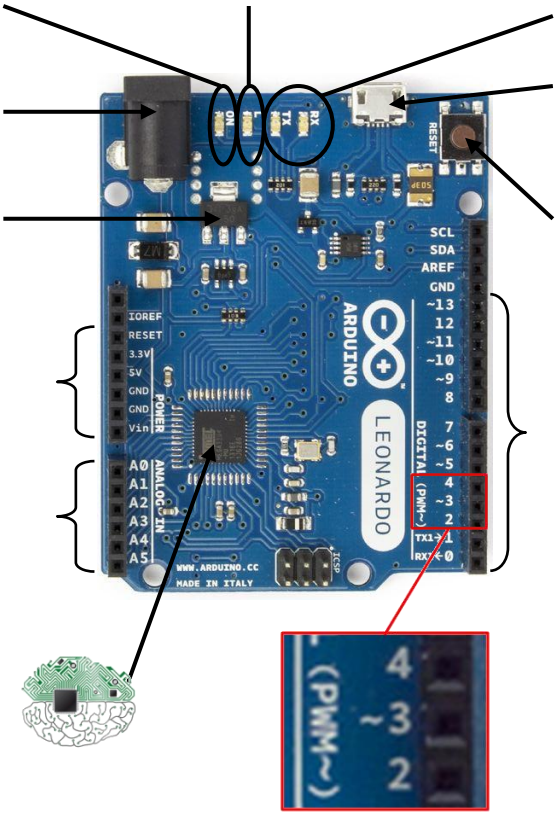
.....

.....

.....

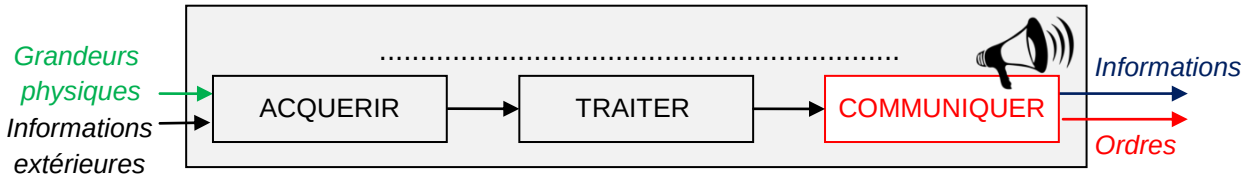
.....

Fonctionement de la carte Arduino Léonardo (que nous allons utiliser cette année) :



Pour fonctionner, et traiter l'information, la carte doit contenir un programme. Nous traiterons la partie programmation ultérieurement.

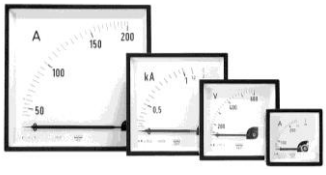
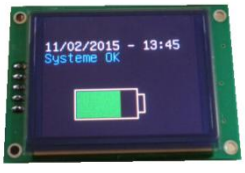
3) Communiquer



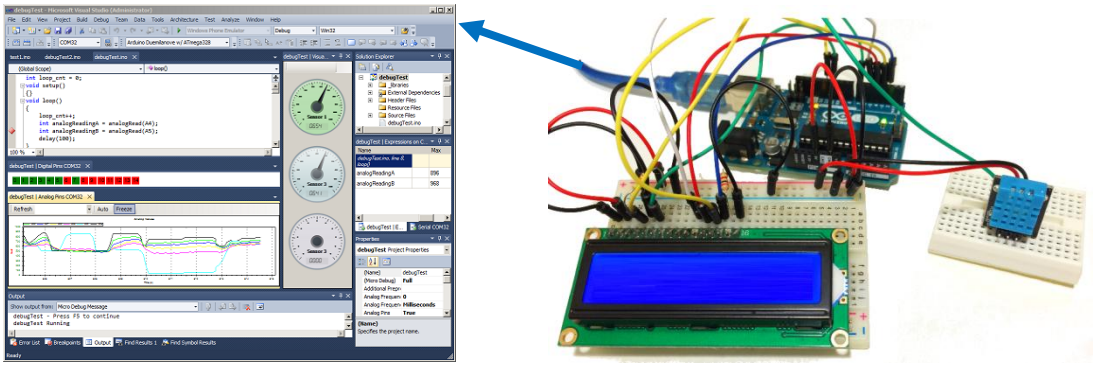
Il y a 2 éléments sortants de la fonction **COMMUNIQUER** :

- Informations : communication avec d'autres systèmes et les interfaces Homme/Machine :

- Dialogue avec l'utilisateur :

Logique (Tout ou rien)	Analogique	Numérique	Graphique
			
Voyant de signalisation LED	Afficheur analogique	Afficheur 7 segments 4 digits	Ecran OLED

- Communication avec l'ordinateur grâce au port USB



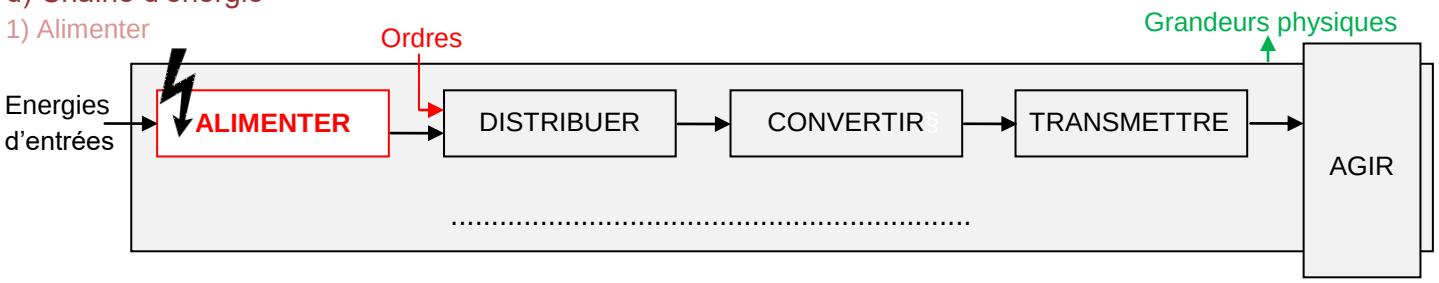
- Communication avec un réseau (grâce à un shield ethernet, wifi ou encore Bluetooth)

- Ordres : communication avec la chaîne d'énergies, pour au final, réaliser une action

Ces ordres peuvent être transmis à la chaîne d'énergie avec ou sans fil (sur les projets Arduino la liaison filaire est très généralement utilisée)

d) Chaîne d'énergie

1) Alimenter



Alimenter c'est fournir au système l'énergie (électrique, pneumatique, hydraulique,…) dont il a besoin pour fonctionner. On distingue 2 types d'énergie :

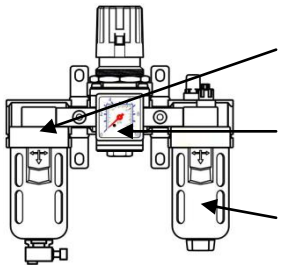
- Energie fournie par un réseau

- Energie électrique

Le réseau public de type courant alternatif, de fréquence 50 Hz, fournit 230V en monophasé ou 400V triphasé. Cette énergie doit être adaptée dans la fonction ALIMENTER, pour fournir une énergie adéquate au système. Pour des projets Arduino, elle est généralement transformée en courant continu et la tension est abaissée entre 6V et 12V à l'aide de transformateur électrique.

- Energie pneumatique

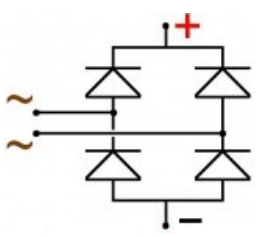
L'énergie pneumatique résulte de la compression de l'air assurée par un compresseur. Cette énergie doit également être adaptée dans la fonction ALIMENTER : Une unité de conditionnement d'air comprimé FRL est généralement placée à l'entrée du système :



- Energie locale

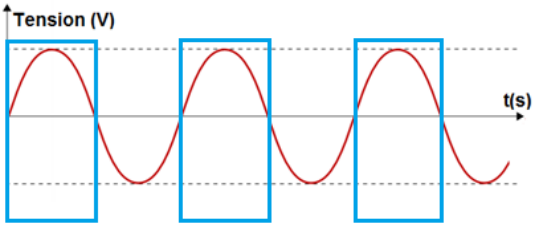
L'énergie électrique est soit produite localement, soit sous une forme directement utilisable, emmagasinée et restituée en fonction des besoins.

Comme précédemment, l'énergie doit également être adaptée dans la fonction ALIMENTER :

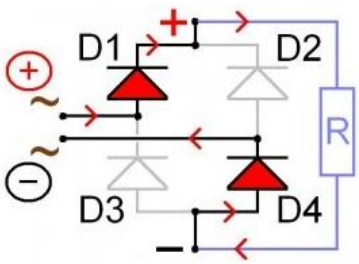


Le **type de courant peut être modifié**. Généralement il est nécessaire de transformer un courant alternatif en courant continue, et non l'inverse. Cela est facilement réalisable à l'aide d'un pont de diodes.

Il est formé de 4 diodes. Il y a deux entrées où doivent arriver la tension alternative et les sorties "+" et "-" où sortent la tension continue.

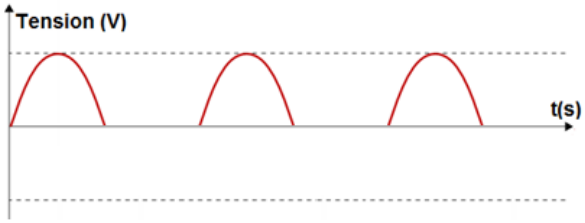


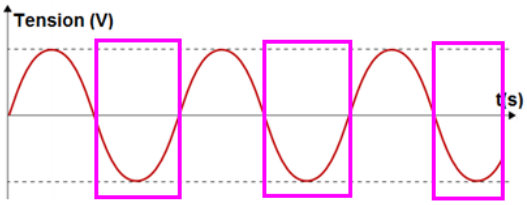
Dans le cas où la tension d'entrée est positive, D1 et D4 sont passantes. D2 et D3 sont bloquées. Le courant traverse la résistance (qui représente la charge) de haut en bas, du "+" vers le "-".



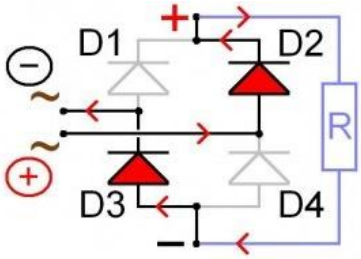
1

Signal à la sortie du pont de diode lorsque D2 et D3 sont bloquées



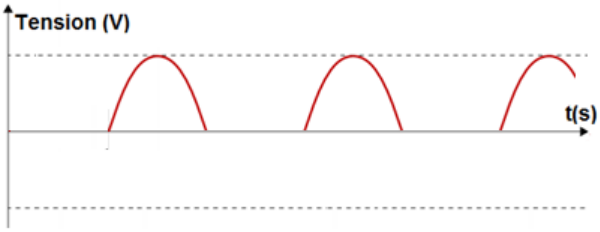


Dans le cas où la tension d'entrée est **négative**, D2 et D3 sont passantes. D1 et D4 sont bloquées. Le courant traverse de même la résistance (qui représente la charge) de haut en bas, du "+" vers le "-".

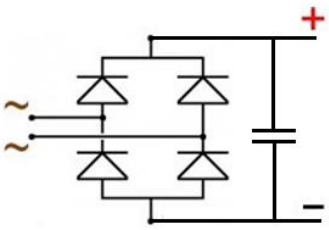


2

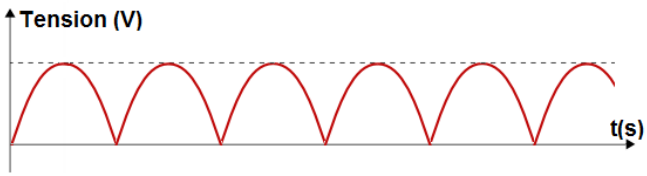
Signal à la sortie du pont de diode lorsque D1 et D4 sont bloquées



Le sens du courant dans la résistance est donc constant, c'est le rôle du pont de diodes. En fait, la tension de sortie n'est pas continue, mais elle est de signe constant. Un condensateur est ajouté entre le "+" et le "-" du pont pour lisser la tension.

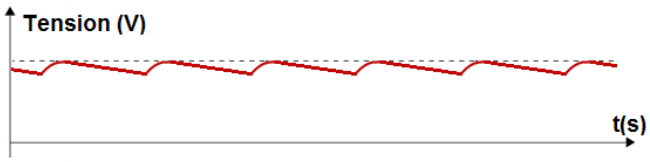


Signal de départ



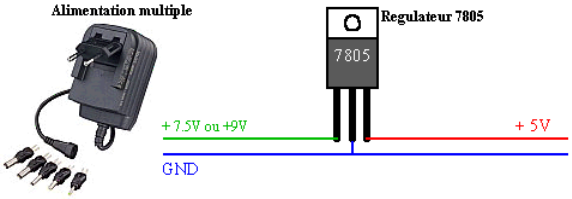
Signal en sortie du pont de diodes

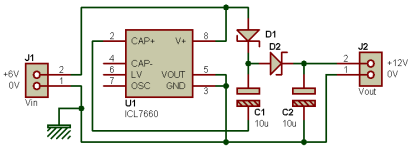
(1 + 2)



Signal en sortie du pont de diodes avec un condensateur

La **tension peut être abaissée** à l'aide d'un régulateur de tension de type **78xx** (comme celui installé par défaut sur l'Arduino). Par exemple un régulateur 7805 permettra d'obtenir une tension de sortie de 5V pour une tension d'entrée au moins supérieure d'un volt (soit au moins 6V à l'entrée). Un régulateur 7812 permettra d'obtenir une tension de sortie de 12V avec une tension d'entrée d'au moins 13V. Etc…





Il est également possible **d'augmenter la tension** mais **avec une intensité très faible** à l'aide par exemple d'un **ICL7660**, de condensateurs et de diodes.

- Piles électrique

Une pile électrique, couramment appelée pile, est un dispositif électrochimique qui convertit l'énergie chimique en énergie électrique grâce à une réaction chimique d'oxydoréduction. Les formats de piles sont couramment désignés par un code normalisé (AA, LR12, CR2032…). Elles fournissent toutes un courant continu, mais leur tension et capacité varie, il est donc important de choisir une (ou plusieurs) pile(s) adaptée(s) au projet.

La capacité d'alimentation est la quantité d'énergie stockée dans une pile. Cette capacité se mesure en wattheures (Wh le symbole). Un Wattheure est la tension (V) que la pile fournit multipliée par la quantité de courant (ampères), que la pile peut fournir sur un certain laps de temps (généralement en heures).

Capacité d'alimentation (Wh) = Tension (V) x Intensité (A) x Durée (H)

Il faut donc additionner l'intensité nécessaire à chaque composant d'un système, la multiplier par sa tension de fonctionnement et par la durée de fonctionnement souhaitée pour dimensionner la ou les pile(s) permettant l'alimentation de l'ensemble.

1 Ah indique en théorie que l'on peut tirer 1 ampère de courant pendant une heure, ou 0,1 A pendant 10 heures, ou 0,01 A (aussi connu comme 10 mA) pendant 100 heures.

- Accumulateurs

Contrairement aux piles, qui une fois « vides » doivent être remplacées, les accumulateurs (aussi appelés batteries) peuvent être rechargés. Comme pour les piles, il existe différents modèles d'accumulateurs, et leur tension et capacité varient. Lors de la conception d'un produit, les ou l'accumulateur(s) doivent être dimensionnés. Il est donc important de choisir un (ou plusieurs) accumulateur(s) adapté(s) au projet.

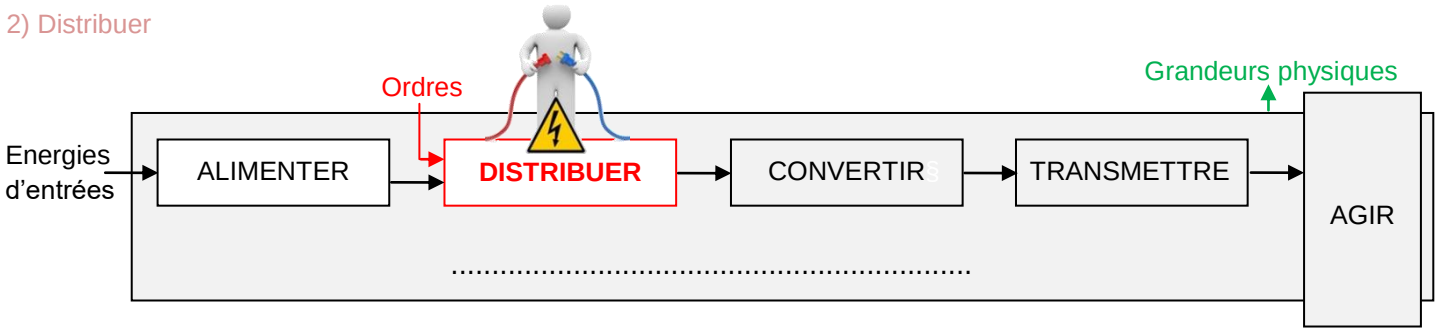
- Panneaux solaires

Ils utilisent l'énergie du soleil. Des cellules photovoltaïques permettent de transformer directement l'énergie solaire en énergie électrique. Ils sont généralement liés à un ou plusieurs accumulateur(s).

- Eoliennes

Un générateur éolien produit de l'électricité à partir de pales orientables. Ces pales ou hélices vont entraîner à leur tour la rotation d'un alternateur qui fournit une puissance électrique liée à la force du vent. Elles sont généralement liées à un ou plusieurs accumulateur(s)

2) Distribuer



C'est sur cette fonction qu'agit la chaîne d'information pour commander le système. Cette fonction permet de délivrer l'énergie au(x) convertisseur(s). L'énergie envoyée par le microcontrôleur (ordres : 5V – **40mA maxi**) permet rarement d'alimenter directement le(s) convertisseur(s). C'est le rôle des préactionneurs comme : le transistor, le réseau de transistor, le pont en H, le relais,…

Seuls des convertisseurs consommant peu d'énergie (comme une LED) peuvent être branchés directement à la sortie de l'Arduino.

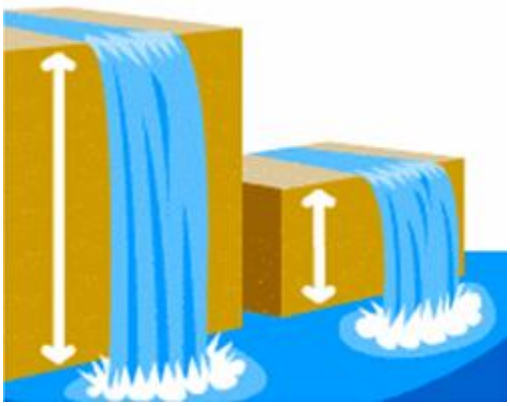
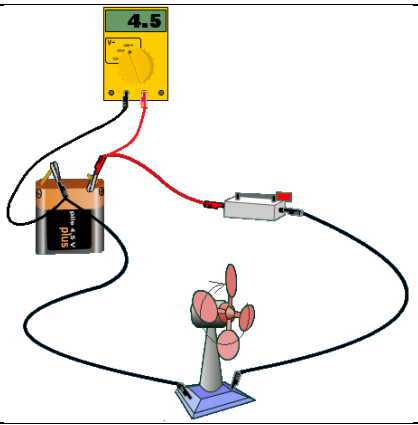
- Branchement sans préactionneur

Sur la sortie d'un Arduino, on peut obtenir **0V** pour un niveau logique bas ou **5V** pour un niveau logique haut. L'intensité maximale pour une sortie est de **40mA**, et de **200mA** sur l'ensemble des sorties.

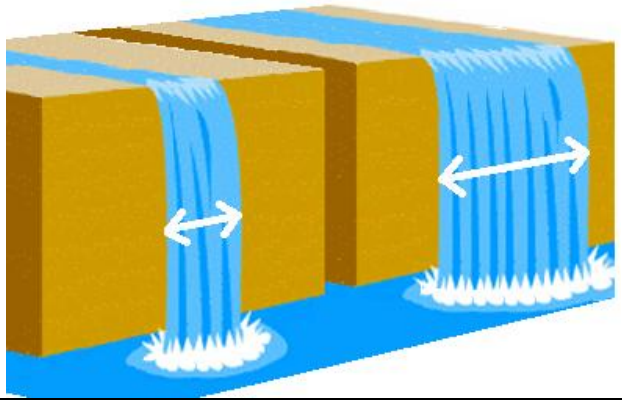
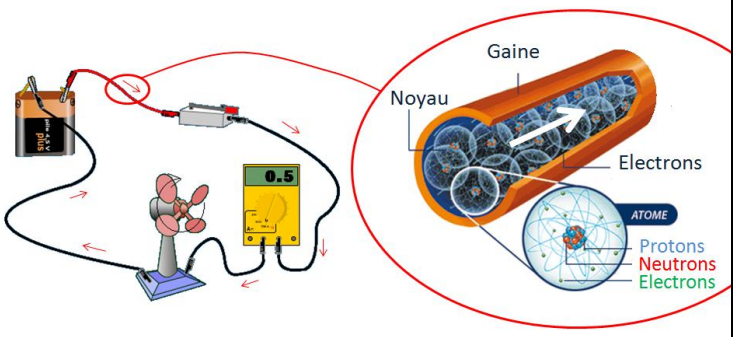
Pour brancher un convertisseur (par exemple une LED) directement sur la sortie d'un Arduino, il faut connaitre et comprendre la loi d'Ohm :

$$U = R \cdot I$$

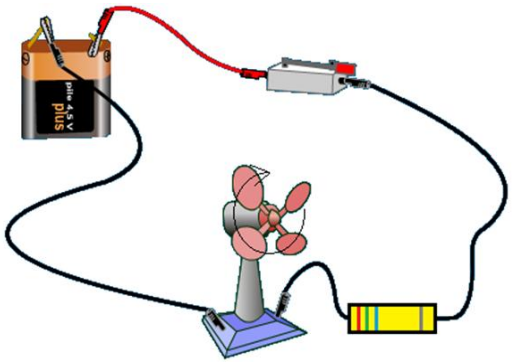
- Qu'est ce que la tension (U) ?

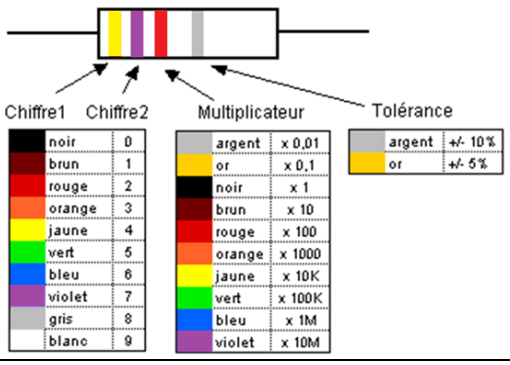
Comparaison à une cascade :	Dans un circuit :
	

- Qu'est que l'intensité (I) ?

Comparaison à une cascade :	Dans un circuit :
	

- Qu'est que la résistance (R) ?

Comparaison à une cascade :	Dans un circuit :
	

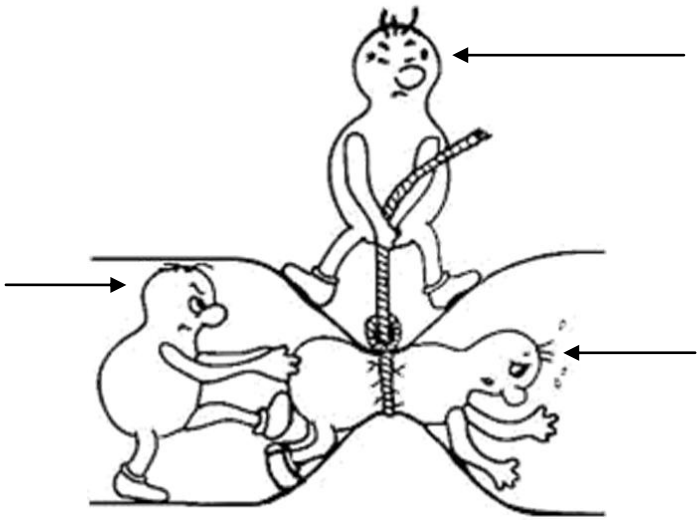


Chiffre1	Chiffre2
noir	0
brun	1
rouge	2
orange	3
jaune	4
vert	5
bleu	6
violet	7
gris	8
blanc	9

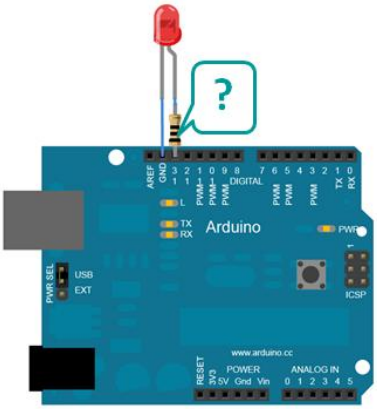
Multiplicateur	
argent	x 0,01
or	x 0,1
noir	x 1
brun	x 10
rouge	x 100
orange	x 1000
jaune	x 10K
vert	x 100K
bleu	x 1M
violet	x 10M

Tolérance	
argent	+/- 10%
or	+/- 5%

- En résumé :



- Branchement d'une LED à la sortie d'un Arduino : application de la loi d'Ohm



Extrait datasheet de la LED

Symbol	Parameter	Device	Typ.	Max.	Units	Test Conditions
λpeak	Peak Wavelength	Super Bright Red	660		nm	I_F=20mA
λD [1]	Dominant Wavelength	Super Bright Red	640		nm	I_F=20mA
Δλ1/2	Spectral Line Half-width	Super Bright Red	20		nm	I_F=20mA
C	Capacitance	Super Bright Red	45		pF	V_F=0V;f=1MHz
V_F [2]	Forward Voltage	Super Bright Red	1.85	2.5	V	I_F=20mA
I_R	Reverse Current	Super Bright Red		10	uA	V_R = 5V

Datasheet : .....

.....

Représentation schématique du problème	Calculs

- Norme CEI 60063

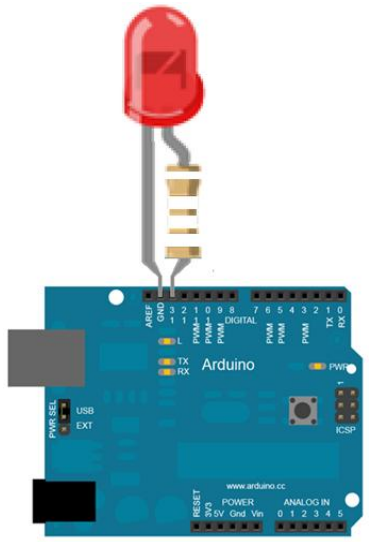
La norme CEI 60063 définit plusieurs séries de valeurs normales pour résistances et condensateurs : E3, E6, E12, E24, E48, E96, E192,...

La **série E24** est utilisée pour le grand public. Les valeurs ci-dessous sont données pour une seule décade, les autres s'obtiennent en multipliant par 10^d, avec d, un entier quelconque

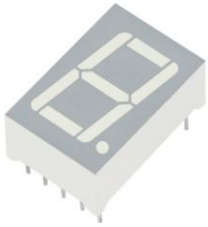
Série E24 :

100, 110, 120, 130, 150, 160, 180, 200, 220, 240, 270, 300, 330, 360, 390, 430, 470, 510, 560, 620, 680, 750, 820, 910

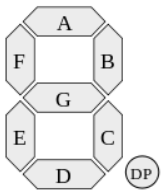
- Choix d'une résistance pour l'exemple page précédente

Représentation	Explications
	

- Branchement d'un afficheur 7 segments



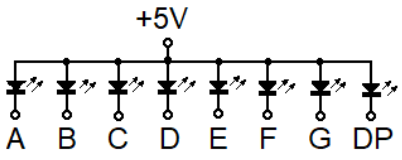
Les afficheurs 7 segments sont utilisés pour afficher les chiffres (bien que quelques lettres soient utilisées pour l'affichage hexadécimal) en allumant ou en éteignant des segments, au nombre de sept. Certains afficheurs comportent un point, servant de séparateur décimal.



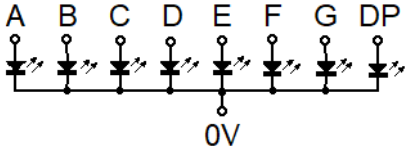
Les segments sont généralement désignés par les lettres allant de A à G. Le point est désigné DP (de l'anglais decimal point). Certains parlent alors dans ce cas d'un afficheur « 8 segments ».

Chaque segment est généralement composé d'une LED (plusieurs parfois sur de gros afficheurs). Deux cas de figures sont présents :

- Afficheur à anodes communes : toutes les anodes sont reliées et connectées au potentiel haut. La commande du segment se fait par sa cathode mise au potentiel bas.



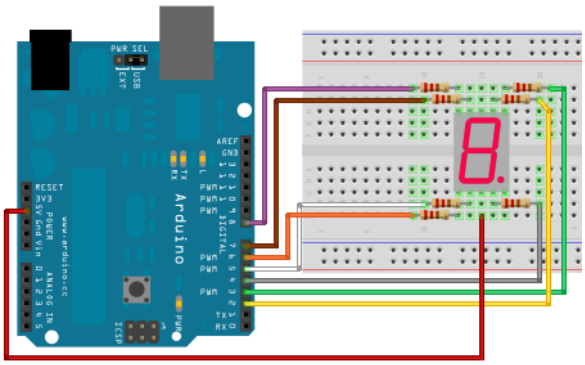
- Afficheur à cathodes communes : toutes les cathodes sont reliées et connectées au potentiel bas. La commande du segment se fait par son anode mise au potentiel haut.



Il y a 2 possibilités pour les relier à l'Arduino :

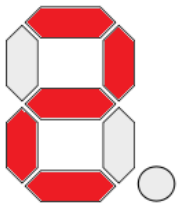
- **Branchement directement à l'Arduino** (sans préactionneur)

Exemple avec un afficheur à anode commune :



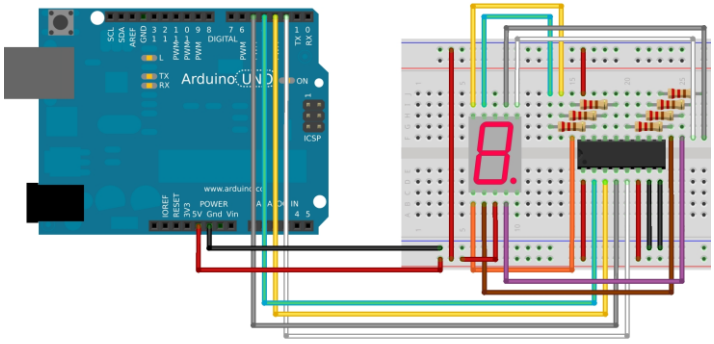
Une résistance est installée entre l'Arduino et chaque segment. Celles-ci ont été dimensionnées à l'aide de la datasheet de l'afficheur.

La programmation de l'Arduino est alors complexe, car chaque segment devra être géré pour chaque chiffre. Par exemple, pour afficher un « 2 » : Allumer : A, B, G, E, D & éteindre : F, C, DP

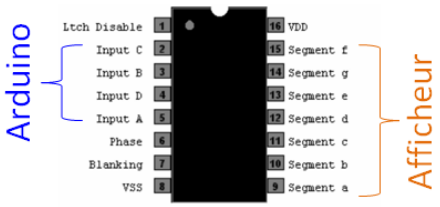


- **Branchement par l'intermédiaire d'un décodeur BCD** (avec préactionneur):

Exemple avec un afficheur à anode commune :



La programmation peut largement être simplifiée grâce à l'utilisation d'un décodeur BCD, interposé entre l'Arduino et le décodeur.
Celui-ci dispose de 7 sorties, notées a,b,c,d,e,f,g correspondant chacune à un des 7 segments de l'afficheur également notés a,b,c,d,e,f,g.
Les entrées sont, au minimum, au nombre de quatre, notées A, B, C et D. Elles représentent le nombre binaire « DCBA » (D étant le bit de poids le plus fort et A celui de poids le plus faible) à afficher.



L'état des sorties du décodeur dépend du nombre binaire que l'on a en entrée. Ce nombre binaire est affiché en décimal sur l'afficheur à 7 segments.

Il faut convertir le nombre à afficher en binaire sur 4 bits (*exemple pour afficher le chiffre 2 : 0010*). Les 0 sont obtenus par un état bas (0V) et les 1 par un état haut (5V). Pour afficher le chiffre 2 il faut donc : input A à 0V, input B à 5V, input C à 0V et input D à 0V.

- Branchement avec préactionneur

- Le transistor

Le transistor le plus simple à utiliser avec un Arduino est le transistor de **type MOSFET**. L'**IRF630**, très couramment utilisé pour l'Arduino, permet le **branchement d'un convertisseur sous une tension maxi de 100V** avec une intensité maximum de 28A (avec radiateur sur le transistor - sans radiateur compter en moyenne 2A maximum).Il existe de très nombreux types de transistors utilisables qui ne sont pas des MOSFET, mais ils semblent moins performants pour ce type d'application (chute de tension, chaleur dégagée).

- Gate : .....

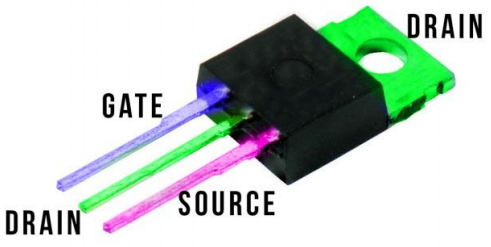
.....

- Drain : .....

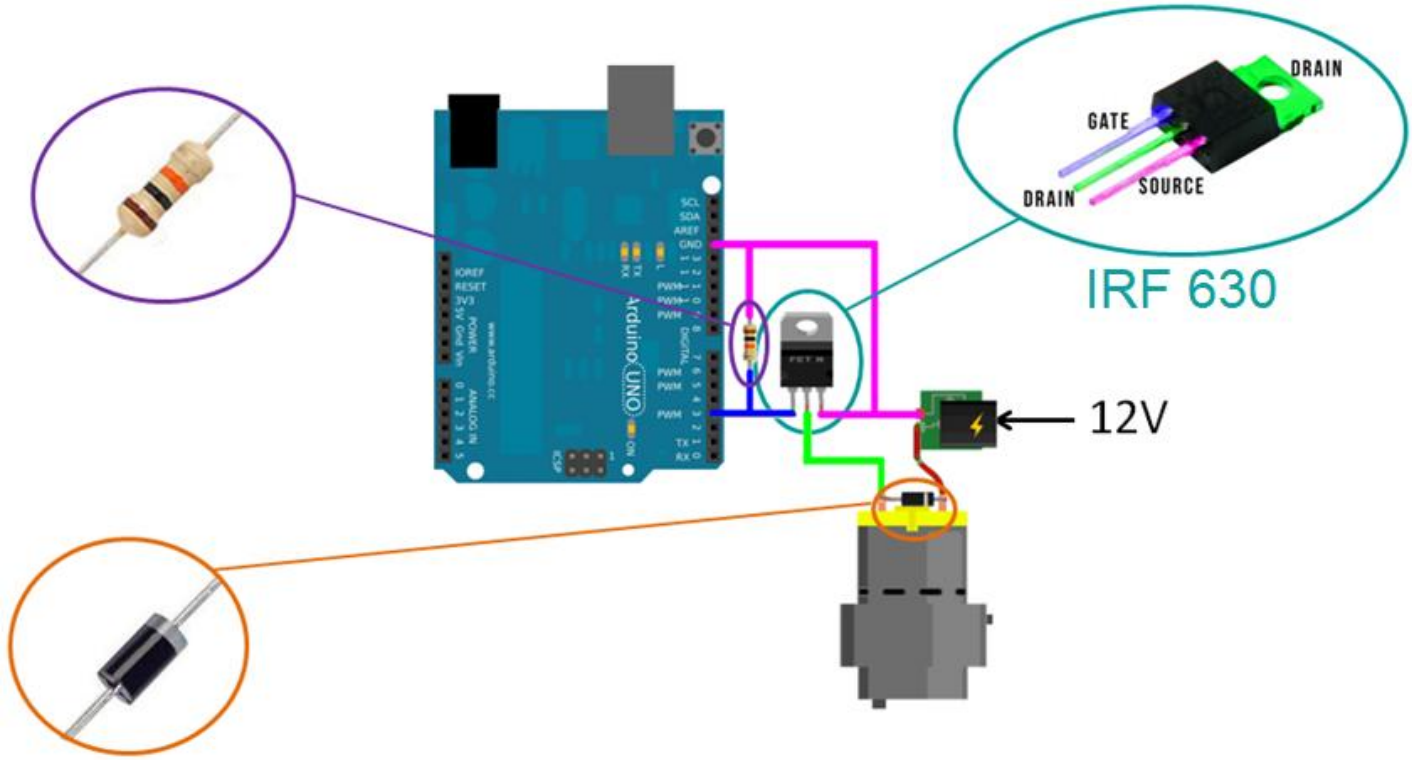
.....

- Source : .....

.....

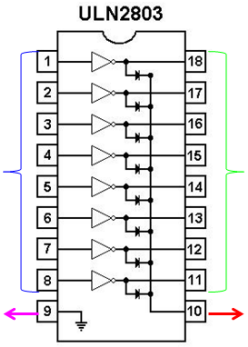


Exemple : IRF630

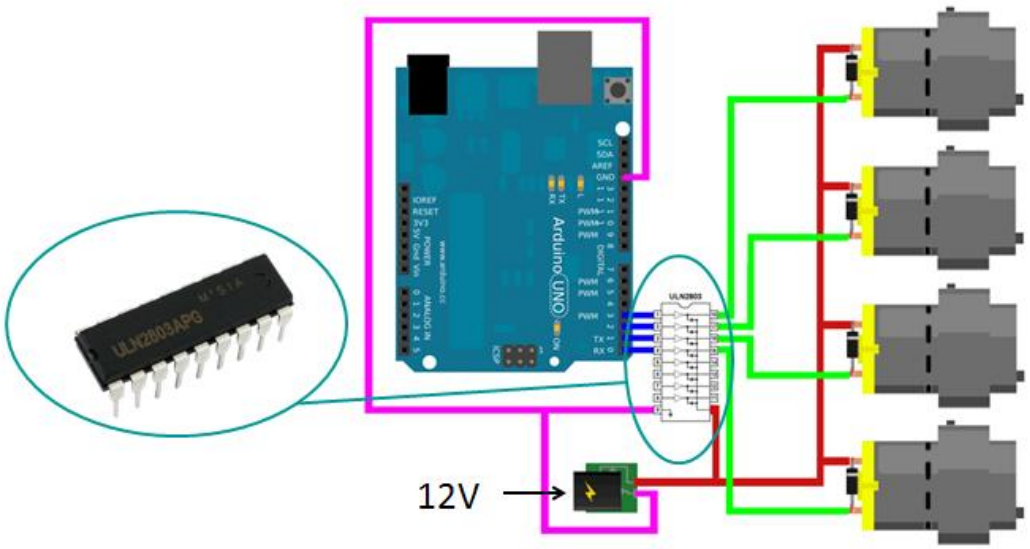


- Réseau de transistors

Eléments intégrant plusieurs transistors, permettant l'**alimentation de différents convertisseurs avec un seul composant**. L'**ULN2803**, très couramment utilisé pour l'Arduino, permet le branchement de 8 convertisseurs sous une tension maxi de 50V avec une intensité maxi de 500mA par sortie.



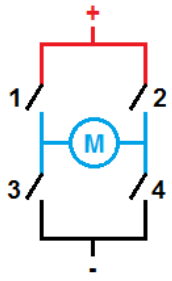
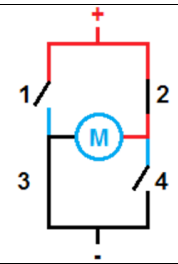
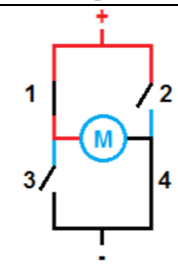
Exemple : ULN2803



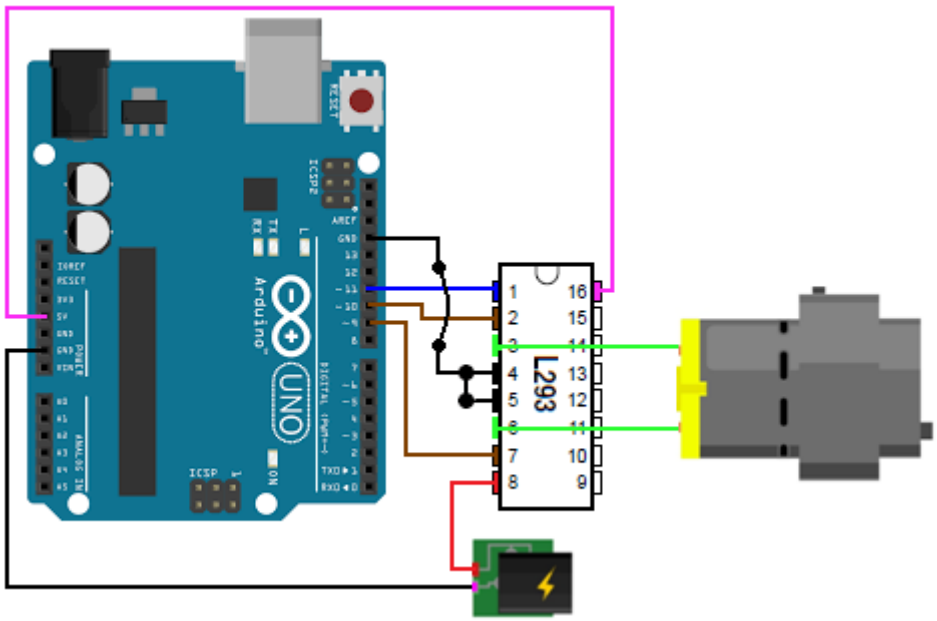
- Pont en H

Les éléments précédents permettent de commander un moteur ou une pompe, mais uniquement dans un sens. Pour **tourner dans les 2 sens**, il faut utiliser un **pont en H** comme le **L293D**. Prévu pour commander des moteurs, il **intègre des diodes de roue libre**.

4 transistors placés en H (d'où son nom) sont commandés 2 par 2 :

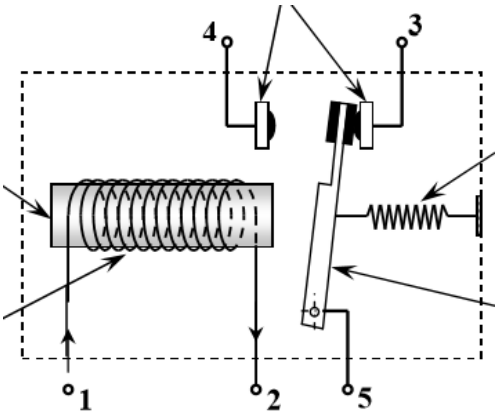
Etat	Schéma	Explications
Repos		
Sens 1		
Sens 2		

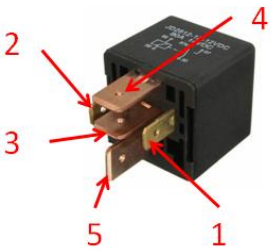
Exemple : L293D

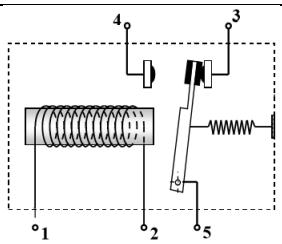
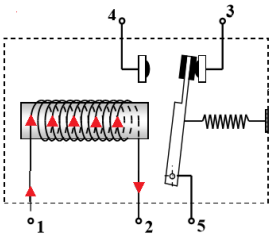
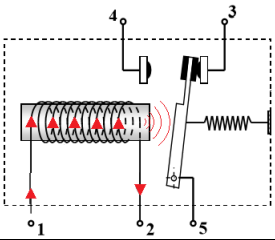
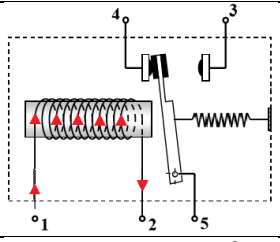
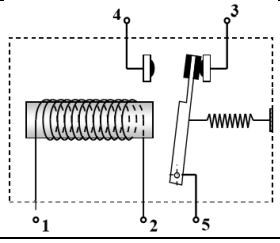


- Relais

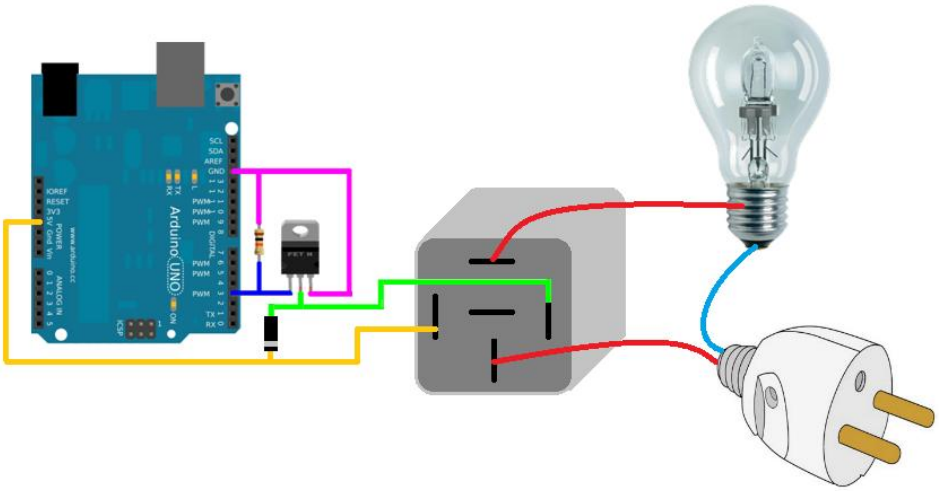
Pour de **fortes tensions**, comme 220V, il est nécessaire d'utiliser un **relais**. Attention, le relais **consomme plus de 40mA, il faut donc le piloter avec un transistor !**





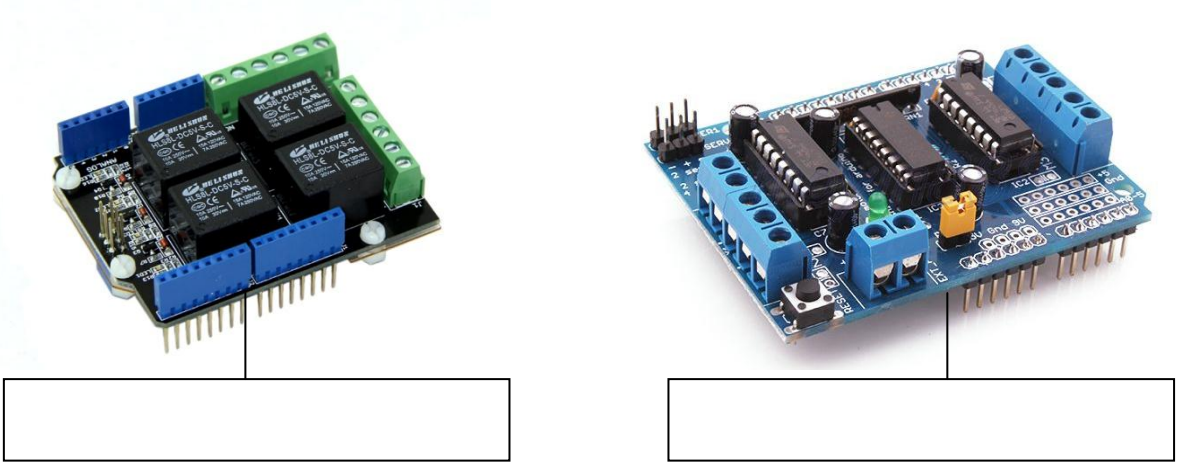
Etat	Schéma	Explications
Non alimenté		
Alimenté		
		
		
Non alimenté		

Exemple : Relais + IRF630

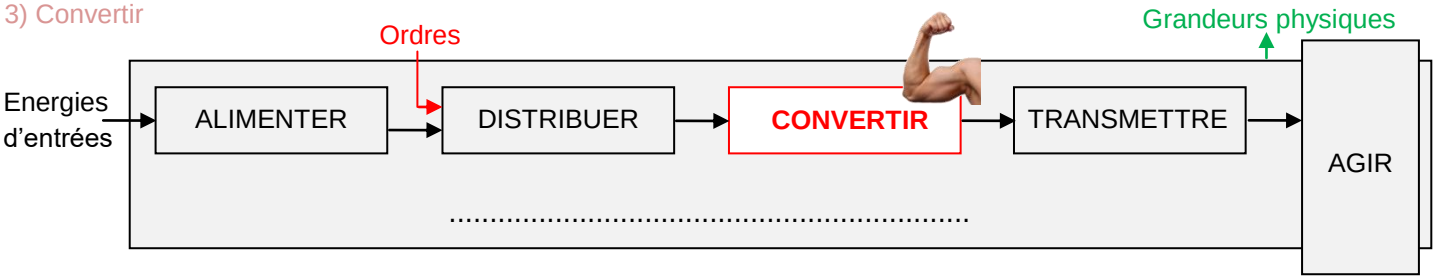


- Shields

Pour plus de simplicité, des shields intégrants directement ces différents préactionneurs existent.



3) Convertir



CONVERTIR : .....

.....

Elle est assurée par les **actionneurs**, encore appelés convertisseurs dans la chaîne d'énergie.

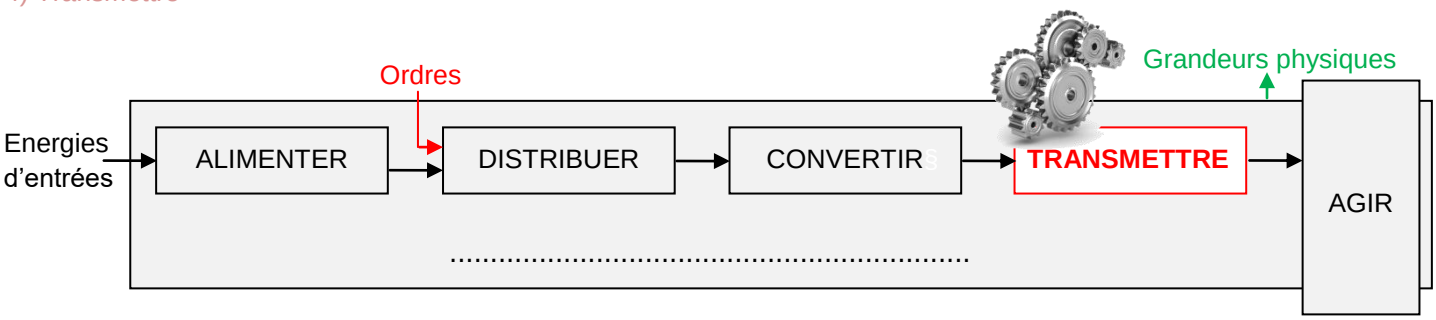
Exemples :

Vérin 12V

Moteur 12V

Pompe 12V

4) Transmettre



TRANSMETTRE : .....

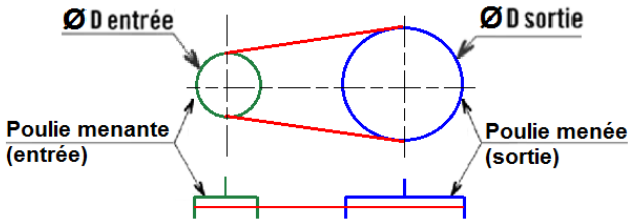
.....

Exemples : transmission mécanique (arbres, réducteurs, embrayages, accouplements, etc.),

RAPPELS :

- Poulies courroie

$$r = \frac{\omega_{sortie}}{\omega_{entrée}} = \frac{D_{entrée}}{D_{sortie}}$$



ω : Vitesse de rotation

D : Diamètre de la poulie

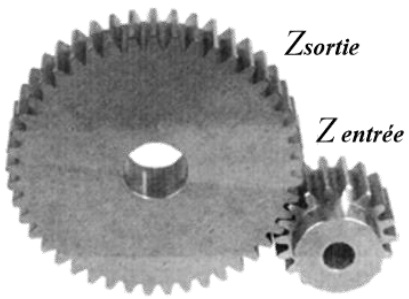
- Engrenages

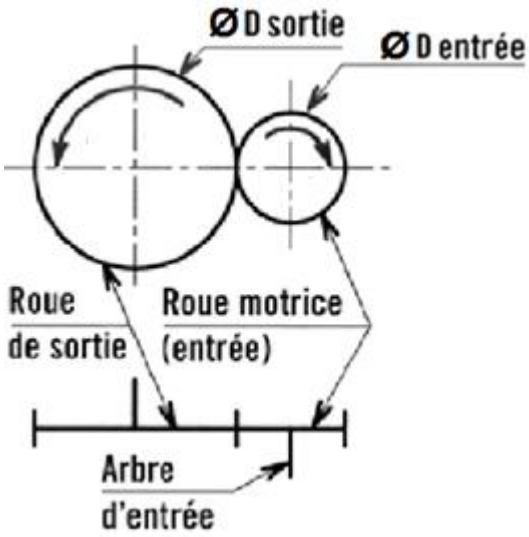
$$r = \frac{\omega_{sortie}}{\omega_{entrée}} = \frac{D_{entrée}}{D_{sortie}} = \frac{Z_{entrée}}{Z_{sortie}}$$

ω : Vitesse de rotation

D : Diamètre primitif

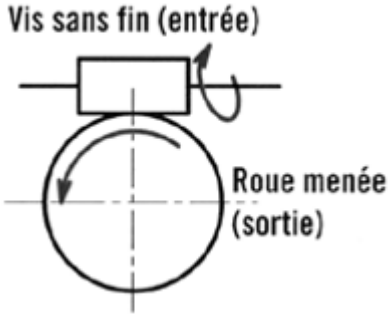
Z : Nombre de dents





- Roue et vis sans fin

$$r = \frac{\omega_{sortie}}{\omega_{entrée}} = \frac{D_{entrée}}{D_{sortie}} = \frac{Z_{entrée}}{Z_{sortie}} = \frac{Z_{vis}}{Z_{roue}}$$

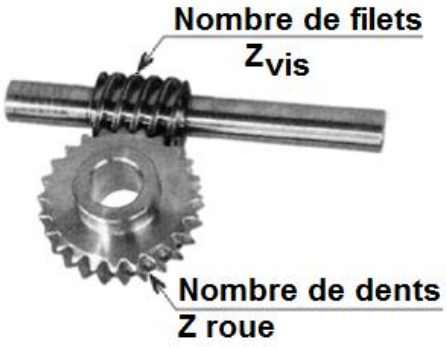


ω : Vitesse de rotation

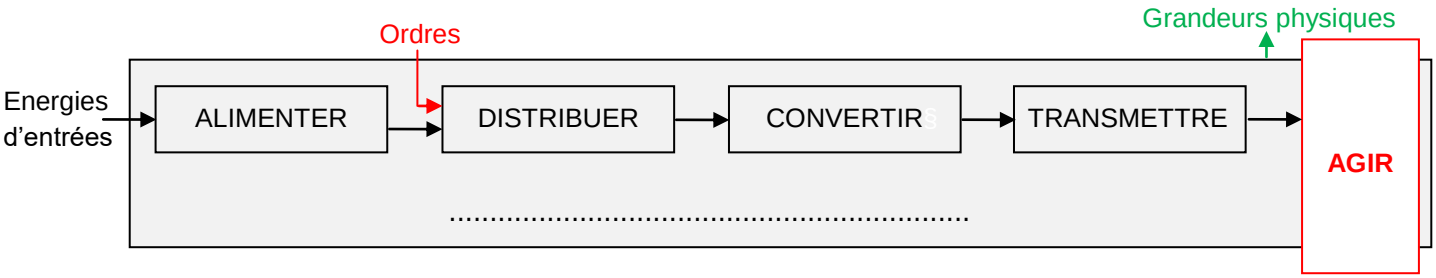
D : Diamètre primitif

Z : Nombre de dents

Z_{vis} : Nombre de filets de la vis



5) Agir



AGIR : .....

.....

Pour programmer notre carte Arduino il faut :

- Une carte Arduino et son câble USB (Type B ou micro en fonction de la carte)
- Un ordinateur avec le logiciel IDE Arduino
- Et **connaitre le langage Arduino** (très proche du C et du C++)

III. Programmation : langage Arduino

Lors de l'observation du fonctionnement d'un objet technique, il y a 2 façons de raisonner : « côté humain » et « côté machine ». Prenons l'exemple avec un robot :

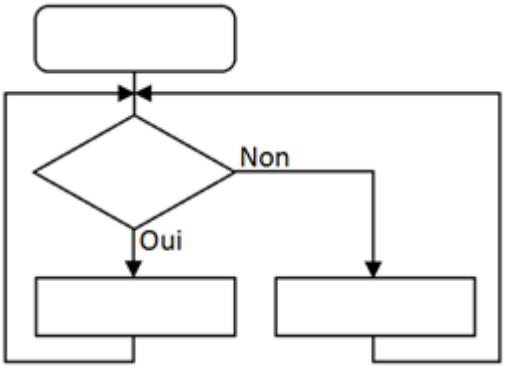
- « côté humain » :.....

- « côté machine » :.....

.....

.....

Pour la réalisation d'un programme, il faudra raisonner « côté machine ». Avant la rédaction d'un programme dans le langage Arduino, il est intéressant de mettre par écrit le fonctionnement final souhaité, soit sous forme de texte comme nous venons de le faire, soit sous forme d'organigramme (voir ci-contre).



a) Syntaxe du langage

La syntaxe d'un langage de programmation est l'ensemble des règles d'écritures liées à ce langage.

1) Le code minimal et structure du programme

Avec l'Arduino, nous devons utiliser un code minimal (même si une partie reste vide) lorsque l'on crée un programme. Ce code permet de diviser le programme que nous allons créer en deux parties.

```
void setup()      //fonction d'initialisation de la carte
{
                  //contenu de l'initialisation
 }

void loop()      //fonction principale, elle se répète (s'exécute) à l'infini
 {
                  //contenu de votre programme
}
```

2) Fonctions

Dans ce code se trouvent deux fonctions. Les fonctions sont en fait des portions de code.

La fonction setup() : .....

.....

.....

.....

La fonction loop() : .....

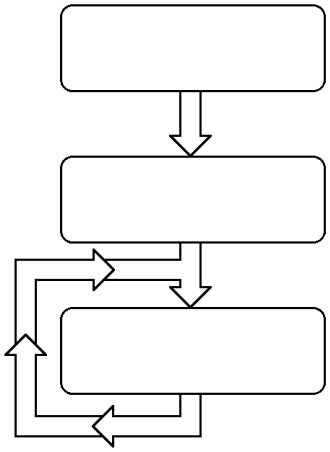
.....

.....

.....

Une troisième partie (non obligatoire) est généralement ajoutée avant le « setup() ». On y indiquera toutes les variables utilisées dans le programme. Cela permettra notamment de donner un nom (variable) à chaque numéro de broche, rendant l'écriture et la lecture du programme beaucoup plus simple.

En schématisant, le programme aura donc cette structure :



3) Les instructions

Les instructions sont des lignes de code qui disent au programme : "fait ceci, fait cela, ...". Se sont les instructions qui vont orchestrer notre programme.

- Les points virgules

Les points virgules terminent les instructions. Si par exemple je dis dans mon programme : "allume la LED" je dois mettre un point virgule après l'appel de cette fonction. **Attention : son oubli est fréquent et rend le programme inutilisable.**

- Les accolades

Les accolades sont les "conteneurs" du code du programme. Elles sont propres aux fonctions, aux conditions et aux boucles. Les instructions du programme sont écrites à l'intérieur de ces accolades.

- Les commentaires

Ce sont des lignes de codes qui seront ignorées par le programme. Elles ne servent en rien lors de l'exécution du programme. **Très importants, ils permettent de relire facilement un programme.**

// cette ligne est un commentaire sur UNE SEULE ligne (2 slashs avant le texte sur une ligne = commentaire)

/*cette ligne est un commentaire sur PLUSIEURS lignes qui sera ignorée par le programme, mais pas par celui qui lit le code */ (1 slash + 1 astérisque avant un texte sur plusieurs lignes et 1 astérisque + 1 slash à la fin de ce texte = commentaire)

- Les accents

Il est formellement interdit de mettre des accents en programmation. Sauf dans les commentaires.

- AVANT LE SETUP : donner un nom aux broches

Pour donner un nom aux broches, nous utiliserons toujours le format suivant :

const int nom = numéro de la broche;

Par exemple, pour une LED branchée sur la broche 2 de l'Arduino, nous allons donner le nom « LEDRouge » (**seulement des caractères alphanumériques : pas d'accents, d'espace,...**) :

```
const int LEDRouge=2; //définition de la broche 2 de la carte en tant que variable : LEDRouge
```

Lorsque nous voudrons utiliser la LED, il suffira d'utiliser la variable *LEDRouge*, et non se rappeler qu'elle est branchée sur la broche numéro 2.

- SETUP : Indiquer si la broche est utilisée en entrée ou sortie : pinMode

Dans le setup() va être défini pour chaque broche si elle est utilisée en entrée ou en sortie. Si aucun nom n'a été donné (comme expliqué à la page précédente) on utilisera le numéro de broche, sinon il faudra utiliser le nom. Pour indiquer si une broche est utilisée en sortie (actionneur), le format suivant sera utilisé :

pinMode(broche, OUTPUT);

Et pour indiquer si une broche est utilisée en entrée (capteur) :

pinMode(broche, INPUT);

Exemple :

```
pinMode(10, OUTPUT);          // Broche numéro 10 utilisée en sortie (actionneur)
pinMode(LEDRouge, OUTPUT);    // Broche nommée « LEDRouge » avant le setup utilisée en sortie (actionneur)
pinMode(A1, INPUT);           // Broche numéro A1 utilisé en entrée (capteur)
pinMode(interrupteur, INPUT); // Broche nommée « interrupteur » avant le setup utilisée en entrée (capteur)
```

- Changer l'état d'une sortie : digitalWrite

Lorsqu'une broche est assignée en sortie, il est alors possible de changer son état : état haut (en fonctionnement) ou état bas (à l'arrêt). Si aucun nom n'a été donné (comme expliqué à la page précédente) on utilisera le numéro de broche, sinon il faudra utiliser le nom.

Pour mettre une sortie à l'état haut (HIGH), le format suivant sera utilisé :

digitalWrite(broche, HIGH);

Et pour mettre une sortie à l'état bas (LOW) :

digitalWrite(broche, LOW);

Exemple :

```
digitalWrite(2, HIGH);          // Mettre la broche numéro 2 à l'état haut
digitalWrite(LEDRouge, HIGH);   // Mettre la sortie « LEDRouge » à l'état haut
digitalWrite(3, LOW);           // Mettre la broche numéro 11 à l'état bas
digitalWrite(moteur, LOW);      // Mettre la sortie « moteur » à l'état bas
```

- Faire une pause : delay

Met en pause le programme pendant une durée donnée : **arrête la lecture du programme durant cette durée**. La durée indiquée est en millisecondes. Le format suivant sera utilisé :

delay (durée en millisecondes);

Exemple :

```
delay (2000);          // Faire une pause de 2 secondes (2000 millisecondes)
```

b) Les variables

Une variable est le **conteneur d'un nombre qui peut changer de valeur**. Ce nombre est stocké dans un espace de la mémoire vive du microcontrôleur, auquel nous donnerons un nom. **Attention : seulement des caractères alphanumériques : pas d'accents, d'espace,...**

La méthode de stockage est semblable à celle utilisée pour ranger des chaussures dans un casier numéroté. On donne un numéro à un casier, dans lequel on peut ranger n'importe quelle paire de chaussures.

1) Définir une variable

Il existe une multitude de nombres : les nombres entiers, les nombres décimaux, ... Il faut donc assigner une variable à un type. Ce type doit être adapté aux nombres à stocker.

Tout comme le casier à chaussures doit être adapté aux types de chaussures : sandales, baskets, bottes,...

Types variables les plus répandus :

Type	Nombre stocké	Valeurs du nombre stocké	Octets utilisés
int		- 32 768 à + 32 767	
long		- 2 147 483 648 à + 2 147 483 647	
char		- 128 à + 127	
flat		- 3,4 x 10^{38} à + 3,4 x 10^{38}	

Exemple, si notre variable "x" ne prend que des valeurs décimales, on utilisera les types int , long , ou char . Si maintenant la variable "x" ne dépasse pas la valeur 87, alors on utilisera le type char..

```
char  x = 0;
```

Attention : l'Arduino à une mémoire limitée. Il ne faut donc pas utiliser uniquement le type « long » sans réfléchir. *C'est comme si nous construisions des casiers pour recevoir des bottes, pour n'y ranger que des sandales : nous utilisons de la place inutilement.*

2) Les variables booléennes

Les variables booléennes sont des variables qui ne peuvent prendre que deux valeurs : ou VRAI ou FAUX. Ce type de variable s'appelle : **boolean**. Elles sont utilisées notamment dans les boucles et les conditions.

Une variable booléenne peut être utilisée de plusieurs manières, mais pour simplifier les choses nous utiliserons la méthode suivante qui est également utilisée pour piloter ou non les sorties de l'Arduino : **LOW** pour mettre à 0 (ou faux) et **HIGH** pour mettre à 1 (ou vrai).

Exemple :

```
boolean v=LOW;  // Création d'une variable booléenne « v » avec en valeur « faux »

v=HIGH; // Passage de la variable booléenne « v » à la valeur « vraie »
```

c) Les opérations simples

La valeur d'une variable peut être modifiée par différents calculs.

1) Additions, soustractions, multiplications et divisions

Calculs les plus simples : addition grâce au signe **+** , soustraction avec le signe **-** , multiplication à l'aide du signe ***** et pour finir le signe ***/*** pour la division.

Exemples :

```
//Définition des variables :
int x = 0; //définition de la variable x avec en valeur 0
int y = 0; //définition de la variable y avec en valeur 0

//Exemples d'additions :
x = 2 + 2; // x vaut donc maintenant 4
y = 2 + x; // y vaut 2 plus la valeur de x (soit 4) donc 6

//Exemples de soustractions :
x = 2 - 1; // x vaut donc maintenant 1
y = y - x; // y vaut la valeur de y (soit 6) moins la valeur de x (soit 1) donc 5

//Exemples de multiplications :
x = 2 * 2; // x vaut donc maintenant 4
y = x * y; // y vaut la valeur de x (soit 4) multipliée par la valeur de y (soit 5) donc 20

//Exemples de divisions :
x = 50 / 5; // x vaut donc maintenant 10
y = y / x; // y vaut la valeur de y (soit 20) divisée par la valeur de x (soit 10) donc 2
```

Attention, pour la division : pour une variable à nombres entiers (char, int ou long) le nombre sera arrondi au nombre inférieur

2) L'incrémentation et la décrémentation

L'incrémentation est l'ajout de la valeur 1 à une variable. Il peut par exemple être réalisé de la façon suivante : x = x +1, mais plus simplement grâce au : **++**

La décrémentation est exactement l'inverse, c'est le retrait de la valeur 1 à une variable. Il peut par exemple être réalisé de la façon suivante : x = x - 1, mais plus simplement grâce au : **--**

Exemple :

```
//Définition des variables :
int x = 0; //définition de la variable x avec en valeur 0
//Exemples d'incrémentation :
x++ ; // x vaut donc maintenant 1 (valeur de départ 0 + 1)
x-- ; // x vaut donc maintenant 0 (valeur précédente 1 - 1)
```

3) L'opération de bascule

Utilisée sur une variable de type booléenne, elle permet d'inverser l'état grâce au signe « ! » qui signifie « not » ou « non ». Correspond à la barre sur la lettre dans certains langages automates (exemple : $\overline{x}$)

```
boolean x=LOW ; // définition de la variable booléenne x avec en valeur 0
x = !x ; // x vaut donc maintenant 1 (l'inverse de la valeur précédente).
```

d) Les conditions

Les conditions servent à tester les variables. Elles sont par exemple utilisées dans un thermostat : si la température dans la pièce est inférieure à la température de consigne il faut allumer le radiateur, sinon le radiateur doit être éteint.

Il faut connaitre les symboles permettant de réaliser ces tests :

Symboles	Signification
==	
<	
<=	
>	
>=	
!=	

Il existe plusieurs conditions :

1) If

On emploie le terme if (de l'anglais "si") pour tester si une variable répond à une condition pour réaliser une ou plusieurs actions. **Dans le cas ou la condition n'est pas remplie, la ou les actions à réaliser seront ignorées.**

Le terme **if** doit être suivi de parenthèses dans lesquelles se trouveront les variables à tester, et d'accolades dans lesquelles seront indiquées la et les actions à réaliser.

Exemple :

```
if (x<20) {                 // Si la valeur de la variable x est inférieur à 20
digitalWrite(2,HIGH);       // Mettre la broche 2 à l'état haut (en fonctionnement) – Si x>20 cette ligne sera ignorée
}
```

2) Else

Il est possible d'ajouter une ou plusieurs actions à réaliser si la condition précédente est fausse. Pour commencer, grâce à la condition **else** (de l'anglais "sinon")

Le terme **else** doit être placé après l'accolade de fermeture de la condition **if**.

Le terme **else** est suivi d'accolades dans lesquelles seront indiquées la et les actions à réaliser dans le cas ou la condition de la fonction **if** précédent la fonction **else** serait fausse.

Exemple :

```
if (x<20) {                 // Si la valeur de la variable x est inférieur à 20
digitalWrite(2,HIGH);       // Mettre la broche 2 à l'état haut (en fonctionnement) – Si x>20 cette ligne sera ignorée
}
else {
digitalWrite(2,LOW);        // Mettre la broche 2 à l'état bas (arrêtée)– Si x<20 cette ligne sera ignorée
}
```

3) Else if

Si la condition n'est pas remplie, il est également possible de tester une autre condition avec la fonction **else if** (de l'anglais "sinon si")

Le terme **else if** doit être placé **après** l'accolade de fermeture de la condition **if**, et **avant** la condition **else** s'il y en a une.

Le terme **else if** doit être suivi de parenthèses dans lesquelles se trouveront les variables à tester, et d'accolades dans lesquelles seront indiquées la et les actions à réaliser. **Dans le cas ou la condition n'est pas remplie, la ou les actions à réaliser seront ignorées.**

Exemple :

```
if (x<20) {                 // Si la valeur de la variable x est inférieur à 20
digitalWrite(1,HIGH);       // Mettre la broche 1 à l'état haut (en fonctionnement). Si x>20 cette ligne sera ignorée
digitalWrite(2,HIGH);       // Mettre la broche 2 à l'état haut (en fonctionnement). Si x>20 cette ligne sera ignorée
}
else if (x=20) {
digitalWrite(1,LOW);        // Mettre la broche 1 à l'état bas (arrêtée). Si x≠20 cette ligne sera ignorée
digitalWrite(2,HIGH);       // Mettre la broche 2 à l'état haut (en fonctionnement). Si x≠20 cette ligne sera ignorée
}
else {
digitalWrite(1,LOW);        // Mettre la broche 1 à l'état bas (arrêtée). Si x<20 cette ligne sera ignorée
digitalWrite(2,LOW);        // Mettre la broche 2 à l'état bas (arrêtée). Si x<20 cette ligne sera ignorée
}
```

e) Les opérateurs logiques

Pour permettre des conditions plus complexes, nous allons utiliser les opérateurs logiques :

Symbole	Opérateur logique
&&	
‖	
!	

1) Et

L'opérateur logique ET, impose que les 2 variables soit vraies (à l'état haut) pour que le résultat soit vrai. Si les 2 résultats ou même seulement l'un d'eux est faux (à l'état bas), le résultat est alors faux :

PERMIERE VARIABLE VRAIE **ET** DEUXIEME VARIABLE VRAIE = RESULTAT VRAIE

Table de vérité **&&** :

VARIABLE 1	VARIABLE 2	SORTIE
0	**0**	
1	**0**	
0	**1**	
1	**1**	

Exemple :

```
if (x<20 && y>32) {     // Si la valeur de la variable x est inférieur à 20 ET la valeur de la variable y est supérieur à 20
digitalWrite(1,HIGH);  // Mettre la broche 1 à l'état haut (en fonctionnement). Si x>20 ou y<32 cette ligne sera ignorée
}
```

2) Ou

L'opérateur logique OU, impose qu'au moins une des 2 variables soit vraies (à l'état haut) pour que le résultat soit vrai. Si les 2 résultats sont faux (à l'état bas), le résultat est alors faux :

PERMIERE VARIABLE VRAIE **OU** DEUXIEME VARIABLE VRAIE = RESULTAT VRAIE

Table de vérité **||** :

VARIABLE 1	VARIABLE 2	SORTIE
0	**0**	
1	**0**	
0	**1**	
1	**1**	

Exemple :

```
if (x<20 || y>32) {     // Si la valeur de la variable x est inférieur à 20 OU la valeur de la variable y est supérieur à 20
digitalWrite(1,HIGH);  // Mettre la broche 1 à l'état haut (en fonctionnement). Si x>20 et y<32 cette ligne sera ignorée
}
```

3) Non

L'opérateur logique NON inverse l'état. Si une variable ou une condition est vraie, elle devient alors fausse, et inversement.

VARIABLE / CONDITION VRAIE = SORTIE FAUSSE

VARIABLE / CONDITION FAUSSE = SORTIE VRAIE

Table de vérité **!** :

VARIABLE / CONDITION	SORTIE
0	
1	

Exemple :

```
if ( !(x<20) {     // Si la valeur de la variable x N'EST PAS inférieur à 20
digitalWrite(1,HIGH);  // Mettre la broche 1 à l'état haut (en fonctionnement). Si x<20 cette ligne sera ignorée
}
```

f) Les boucles

En programmation, une boucle est une instruction qui permet de répéter un programme entier ou une partie de programme. Il existe principalement deux types de boucles :

La boucle conditionnelle, qui teste une condition et qui exécute les instructions qu'elle contient tant que la condition testée est vraie.

La boucle de répétition, qui exécute les instructions qu'elle contient, un nombre de fois prédéterminé.

1) while (boucle conditionnelle)

La boucle while de l'Anglais « tant que » va permettre de répéter une ou plusieurs actions TANT QUE la condition est respectée.

Lorsque la condition n'est plus remplie, le microcontrôleur sort de la boucle et continue la lecture du programme à la suite.

Si la condition n'est pas remplie dès le départ, on ne rentre même pas dans la boucle et on passe directement à la suite : le programme contenue dans la boucle ne sera alors pas lu.

Le terme **while** doit être suivi de parenthèses dans lesquelles se trouveront les variables à tester, et d'accolades dans lesquelles seront indiquées la et les actions à réaliser.

Exemple :

```
int compteur = 0; //variable compteur qui va stocker le nombre de fois que la boucle aura été exécutée

while(compteur != 3) {   // Tant que compteur est différent de 3, on boucle
compteur++;  // On incrémente la variable compteur à chaque tour de boucle
}
```

2) do…while (boucle conditionnelle)

La boucle do...while de l'Anglais « Faire... tant que » va permettre de FAIRE une ou plusieurs actions TANT QUE la condition est respectée.

A la différence de la précédente, on rentre dans la boucle quelque soit l'état de la condition, puis on teste la condition une fois la boucle terminée, pour savoir si on relance la boucle ou si on en sort.

Donc si la condition n'est pas remplie dès le départ, on rentre quand même la boucle pour l'exécuter au moins une fois.

Le terme **do** doit être suivi d'accolades dans lesquelles seront indiquées la et les actions à réaliser. Après l'accolade de fermeture sera placé le therme **while** suivi de parenthèses dans lesquelles se trouveront les variables à tester.

Exemple :

```
int compteur = 3; //variable compteur qui va stocker le nombre de fois que la boucle aura été exécutée

do {
compteur++;  // On incrémente la variable compteur à chaque tour de boucle
}while(compteur < 3);  // Tant que compteur est inférieur à 3, on boucle
```

3) for (boucle de répétition)

La boucle **for** de l'Anglais « pour », va exécuter la boucle **pour** les paramètres donnés entre parenthèses. Il y a 3 paramètres séparés par des points-virgules :

- Création d'une variable avec une valeur de départ (*exemple : int compteur = 0*)
- Condition à tester sur cette variable (*exemple : compteur < 3*)
- Instruction à exécuter sur cette variable (*exemple : compteur++*)

L'ensemble suivi d'accolades dans lesquelles seront indiquées la et les actions à réaliser.

Exemple :

```
for(int compteur = 0; compteur < 3; compteur++)
/* création d'une variable « compteur » avec comme valeur de départ 0, la boucle est exécutée tant que compteur est inférieur à 3, avec une incrémentation à chaque lecture de la boucle */
{
digitalWrite(1,HIGH);        // Mettre en fonctionnement la broche 1
delay(1000);                 // Attendre 1 seconde
digitalWrite(1, LOW);        // Arrêter le fonctionnement de la broche 1
delay(1000);                 // Attendre 1 seconde
}
```

4) Cas particulier : la boucle infinie

Il peut être utile dans certains cas de réaliser une boucle infinie. Il suffit alors de créer une boucle **while** et de lui assigner le chiffre 1 (qui signifie que la condition est toujours vraie).

Exemple :

```
while(1)  // Tant que « vrai » on boucle  (donc on boucle à l'infini)

{
digitalWrite(1,HIGH);        // Mettre en fonctionnement la broche 1
delay(1000);                 // Attendre 1 seconde
digitalWrite(1, LOW);        // Arrêter le fonctionnement de la broche 1
delay(1000);                 // Attendre 1 seconde
}
```

La fonction loop() se comporte comme une boucle infinie, car elle se répète après avoir fini d'exécuter ses tâches.

g) Les fonctions

Une fonction est un fragment de programme. Pour ne pas écrire plusieurs fois les mêmes instructions dans un programme, nous allons le décomposer en fonctions, qui pourront ensuite être appelées au moment désiré. De plus, cela permet de simplifier la lecture.

En fait, lorsqu' on programme un Arduino, chaque partie du programme doit être dans une fonction. Vous en connaissez d'ailleurs déjà 2 qui sont obligatoires : **setup()** et **loop()**.

Beaucoup de fonctions existent déjà dans le langage Arduino, il suffit de les appeler dans le programme. Lorsque vous réalisez une pause dans un programme, vous écrivez **delay(1000)** : c'est une fonction !

*Par exemple dans le programme d'un portail automatique, nous pouvons décomposer le programme en une partie principale **loop()** qui va interroger les différents capteurs (télécommande, barrière infrarouge,...) et appeler une fonction **ouvrir()** ou une fonction **fermer()** dans lesquelles l'ensemble des instructions permettant d'ouvrir ou de fermer le portail seront présentes.*

Pour créer une fonction il faut indiquer 2 caractérisitques :

- Son **nom**
- Avec/sans **paramêtre(s)**

La fonction doit être **suivie d'accolades** dans lesquelles seront indiquées la et les actions à réaliser.

1) Nom d'une fonction

Il est choisi par la personne qui écrit le programme, mais doit être le plus explicite possible pour garantir une compréhension rapide du programme. Attention, ni espace, ni caractères spéciaux, et le minuscules et majuscules sont prises en compte.

Exemple sans paramètre ni valeur de retour :

Création d'une fonction permettant d'allumer 3 LED branchées sur les broches 2, 3 et 4 de l'Arduino (Exemple de nom de fonction permettant d'allumer 3 LED : **AllumerLes3LED**) :

```
void AllumerLes3LED() {   // Création d'une fonction « AllumerLes3LED » sans paramètre ni valeur de retour
digitalWrite(2,HIGH);     // Mettre en fonctionnement la broche 2
digitalWrite(3,HIGH);     // Mettre en fonctionnement la broche 3
digitalWrite(4,HIGH);     // Mettre en fonctionnement la broche 4
}
```

Pour appeler cette fonction dans mon programme, je devrais alors écrire :

```
AllumerLes3LED();
```

2) Paramètres

Certaines fonctions ont besoin de recevoir des valeurs pour leur execution : ce sont **les paramètres de la fonction**.

Par exemple lorsqu' on appelle la fonction « **delay()** » nous indiquons une valeur correspondant au temps de pause en millisecondes : **delay (1000);**

Le ou les paramètres sont indiqués entre les paranthèses qui suivent le nom de la fonction. Certaines fonctions n'ont pas de paramètres, les paranthèses restent alors vides (exemple : la fonction **loop()**).

Certaines fonctions revoient même des valeurs après leur exection : se sont **les valeurs de retour** *(comme par exemple une fonction qui permettrait de calculer une distance avec un obstacle, revoyait la distance mesurée une fois la fonction exectuée)* .

Nous ne traiterons pas de telles fonctions cette année. Pour indiquer qu'une fonction ne renvoie aucune valeur, elle est précédée du mot **void** : comme les fonction setup() et loop().

Donc **cette année, lorsque vous créerez une fonction, elle sera toujours précédée de « void ».**

Exemple avec un paramètre sans valeur de retour :

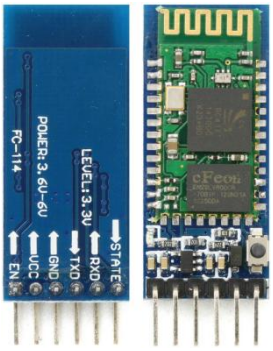
Création d'une fonction « test » permettant d'allumer deux LED branchées sur les broches 2 et 3 de l'Arduino si le paramètre est >5, de les éteindre s'il est <5, ou d'allumer celle de la broche 2 et d'éteindre celle de la broche 3 s'il est =5. Le paramètre est une variable nombre de type char.

```
void test(char nombre) { // Création d'une fonction « test » avec un paramètre « nombre » de type char sans valeur de retour

if (nombre>5) {            // Si le paramètre « nombre » est supérieur à 5
  digitalWrite(2,HIGH);    // Mettre en fonctionnement la broche 2
  digitalWrite(3,HIGH);    // Mettre en fonctionnement la broche 3
}

else if (nombre<5) {       // Sinon si le paramètre « nombre » est inférieur à 5
  digitalWrite(2,LOW);     // Mettre à l'arrêt la broche 2
  digitalWrite(3,LOW);     // Mettre à l'arrêt la broche 3
}

else {                     // Sinon (donc égal à 5)
  digitalWrite(2,HIGH);    // Mettre en fonctionnement la broche 2
  digitalWrite(3,LOW);     // Mettre à l'arrêt la broche 3
}
```

Pour appeler cette fonction dans mon programme avec « 4 » comme paramètre, je devrais alors écrire :

```
test(4);
```

h) Liaison Bluetooth



Les modules Bluetooth, comme le HC-05 ou HC-06 permettent une communication sans fil entre un port série de la carte et un ordinateur, ou encore un téléphone portable. Sur l'arduino le port série par défaut se trouve sur les boches 0 et 1 de la carte, mais sur certaines cartes il est également utilisé pour téléverser des programmes dans la carte (uno par exemple). Il est donc judicieux de brancher les broches de transmission (Tx) et de récépetion de données (Rx) du module à n'importe quels autres pins analogiques pour garder la voie série libre, en émulant un port série sur ces pins. Pour ce faire il suffit d'inclure la bibliothèque « SoftwareSerial » :

Croquis → Inclure une bibliothèque → SoftwareSerial ou ajouter la ligne :

```
#include <SoftwareSerial.h>
```

Esuite il faut ajouter la ligne pour donner un nom au port série émulé (par exmple : bluetooth) et indiquer les broches Rx et Tx que l'on souhaite créer (remplacer Rx et Tx par leur numéro de broche).

Attention la broche Tx du module doit être branchée à une broche Rx de l'Arduino et la la broche Rx du module doit être branchée à une broche Tx de l'Arduino. En effet, la boche qui parle d'un côté doit être écouté de l'autre.

```
SoftwareSerial bluetooth (Rx, Tx);
```

Et créer une variable de type char dans laquelle sera enregistrée la commande envoyée en Bluetooth

```
char commande;
```

Pour finir, le module fonctionnant par défaut en 9600 bauds, il faut ajouter cette ligne dans le setup () :

```
bluetooth.begin(9600);
```

Le module est maintenant opérationnel, il suffit d'enregistrer les informations envoyées par les commandes :

```
if (bluetooth.available())  {                    // Si le module Bluetooth transmet une information
  commande = bluetooth.read();    }              // Lire l'information et l'enregistrer dans la variable "commande"
```

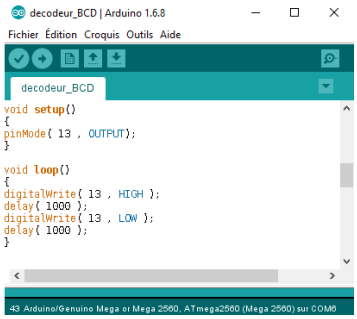
En fonction de la valeur de **commande** il n'y pas plus qu'a réaliser les actions voulues.

L'envoi d'ordres en Bluetooth peut être fait d'un téléphone portable avec, par exemples, l'application « Bluetooth Terminal », « Bluetooth Electronics » ou encore en ayant préalablement créé une petite application avec « App Inventor »

V. Programmation : Logiciel IDE Arduino

Il existe plusieurs logiciels permettant la programmation Arduino, notamment « Ardublock » qui est très connu pour sa programmation simplifiée sous forme de blocs à assembler tel un puzzle. Pour notre part, nous utiliserons le logiciel open-source **Arduino officiel nommé IDE Arduino**. Il peut être installé sous Windows, Linux et Mac Os.

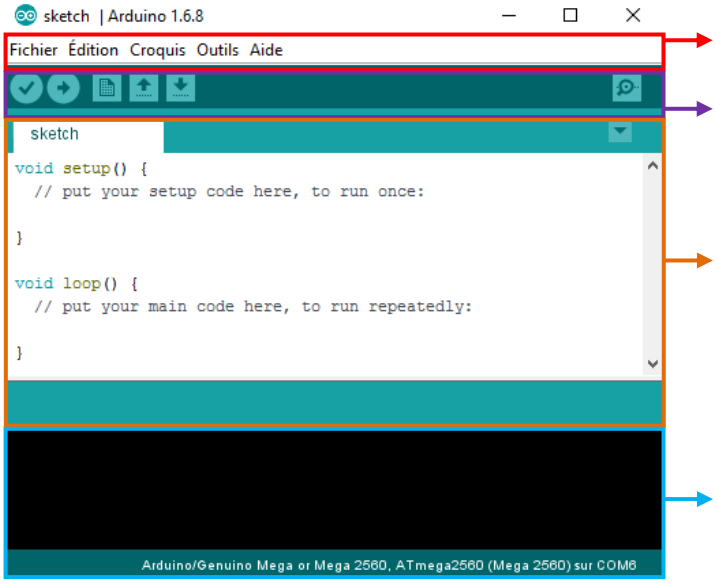
Ardublock



IDE Arduino

a) Présentation du logiciel

IDE Arduino a une interface simple, claire et intuitive que l'on peut découper en 4 parties :



Les parties importantes du **Menu :**

- Fichier → Exemples : Programmes existants qui peuvent servir de base pour tester du matériel ou même faire un programme.
- Outils → Moniteur série : Permet d'afficher des informations envoyées par l'Arduino (voir plus loin).

- Outils → Type de carte : Sélectionner la carte utilisée *(exemple : Uno, Léonardo,...)*. A sélectionner avant de rédiger le programme car des erreurs peuvent apparaitre si la carte sélectionnée n'est pas la bonne *(exemple : erreur car le port demandé n'existe pas sur la carte sélectionnée)*.

- Outils → Port : Le branchement d'une carte Arduino crée un port COM virtuel. Il faut ensuite ce port COM pour que le la communication entre la carte Arduino et le logiciel soit possible.

- Outils → Formatage automatique : Décale automatiquement le texte à droit pour chaque nouvelle accolade : **permet de contrôler très rapidement les oublies d'accolades.**

Barre d'action :

	Vérifier : Vérifie la cohérence du programme, et indique les erreurs sur l'écran de **débogage**
	Téléverser : Envoie le programme sur la carte Arduino
	Nouveau : Commence un nouveau programme
	Ouvrir : Ouvre un programme précédemment enregistré
	Enregistrer : Enregistre le programme. Attention à bien enregistrer dès le début. **Favoriser « Fichier → Enregistrer sous » et sauvegarder sous différents noms à plusieurs périodes de la programmation.** Cela permet d'avoir des versions avec les différentes modifications apportées.
	Moniteur série : Ouvre un terminal série permettant la communication avec l'Arduino : envoi d'ordres ou récupération d'informations (comme l'état de capteurs)

Fenêtre de programmation :

C'est ici que vous rédigez votre programme. Les instructions reconnues par le logiciel apparaissent en couleur.

Débogueur :

A surveillez lors de la vérification ou du téléversement. C'est lui qui indique le bon déroulement des opérations ou les erreurs.

b) Téléverser un programme

- Tester le matériel :

Avant d'envoyer un nouveau programme il est intéressant de tester le bon fonctionnement du matériel utilisé. Pour cela nous allons utiliser un programme se trouvant dans les exemples, qui permet de faire clignoter la LED intégrée à la carte Arduino (reliée au port 13 de la carte).

Relie la carte Arduino à l'ordinateur **via un concentrateur USB**

Ouvre le programme d'essai : **Fichier → Exemples → 01.Basics → Blink**

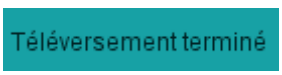
Sélectionne le type de la carte à programmer : **Outils → Type de carte → Sélectionner la carte utilisée**

Sélectionne le port COM de la carte : **Outils → Port → COM (nom de la carte)**

Téléverse le programme : 

Le logiciel indique alors : 

Puis : 

Et enfin : 

En cas d'erreur, vérifie que tu as bien réalisé toutes les opérations listées précédemment.

Il est possible de réaliser son propre programme de vérification, pour tester davantage de matériel que la carte Arduino. Par exemple mettre en fonctionnement tous les actionneurs, ou encore faire réagir un actionneur en fonction de l'état d'un capteur.

- Vérifier le programme :

Il faut ensuite ouvrir le programme final et le vérifier (). En cas d'erreur il faut commencer par vérifier qu'il ne manque aucun point-virgule (première cause d'erreur).

- Téléverser :

Relie la carte Arduino à l'ordinateur **via un concentrateur USB**

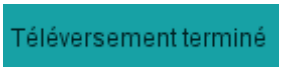
Sélectionne le type de la carte à programmer : **Outils → Type de carte → Sélectionner la carte utilisée**

Sélectionne le port COM de la carte : **Outils → Port → COM (nom de la carte)**

Téléverse le programme : 

Le logiciel indique alors : 

Puis : 

Et enfin : 

En cas d'erreur, vérifie que tu as bien réalisé toutes les opérations listées précédemment.

Il ne reste alors plus qu'à tester le bon fonctionnement du programme sur la carte.

c) Terminal série

Le terminal série est utilisé pour interagir avec la carte Arduino pendant l'exécution d'un programme. Cela permet notamment d'envoyer des ordres pour réaliser des actions ou de récupérer des informations (comme l'état de capteurs).

La fonction qui nous sera utile cette année est la récupération de l'état d'un capteur et d'un texte expliquant l'élément envoyé. Pour ce faire il y a quelques lignes à intégrer dans le programme :

- Dans le setup, il faut démarrer le port série à la vitesse de 9600bits/sec (vitesse standard) :

```
void setup() {
Serial.begin(9600); // A intégrer dans le setup pour démarrer le port série à la vitesse de 9600bits/sec (vitesse standard)
}
```

- Dans le programme il faut alors utiliser les fonctions d'envoi sur le port série :

Serial.print() qui va envoyer l'élément entre parenthèses OU Serial.println() qui va envoyer l'élément entre parenthèses et faire un retour à la ligne ensuite.

- Et indiquer dans les parenthèses l'élément à envoyer :

Pour envoyer un texte, il faudra l'indiquer entre guillemets, par exemple : *"Donnée du capteur de température :"*

Attention : Les chaînes de caractères sont toujours définies à l'intérieur de guillements doubles (**"ABC"**) et les caractères sont toujours définis à l'intérieur de guillemets simples (**'A'**).

Pour envoyer la valeur d'une variable il suffit de nommer la variable (sans guillemets), par exemple : *temperature*

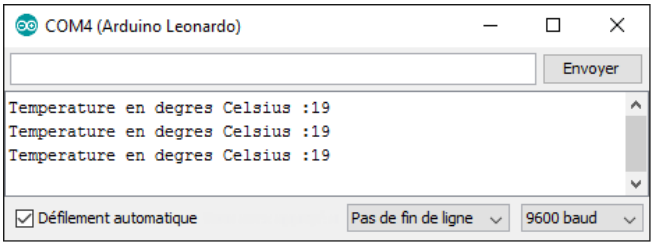
```
void loop() {
Serial.print("texte à envoyer");   // Envoie le texte entre guillemets ou la valeur de la variable indiquée
//ou
Serial.println(variable);          // Même fonction, mais avec un retour à la ligne à la fin du message
```

Une fois le programme téléversé, il n'y a plus qu'à ouvrir le terminal série pour recevoir les informations  .

Exemple de programme permettant de relever une température :

```
// définition des broches utilisées
const int temperature = A1;   // Capteur de température sur la broche A1
int valeurLue = 0;            // Création d'une variable pour stocker la température mesurée avec comme valeur intiale 0

void setup() {
pinMode(temperature, INPUT);      // Utilisation de la broche du capteur de température en entrée
Serial.begin(9600);               // Démarrage du port série à la vitesse de 9600bits/sec (vitesse standard)
}

void loop() {
valeurLue = analogRead(temperature) ;          // Copie de la valeur donnée par le capteur dans la variable « temperature »
Serial.print("Temperature en degres Celsius :"); // Envoi du texte « Température en degrés Celsius »
Serial.println(valeurLue) ;                    // Envoi de la valeur de la variable « temperature »
delay(1000);                                   // Attente d'une seconde
}
```

Téléversement , puis ouverture du terminal série pour recevoir les informations .

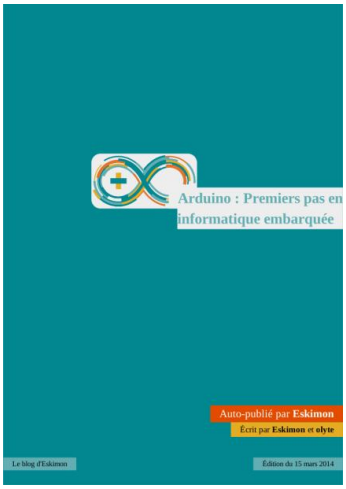


VI. Pour aller plus loin :

- **Arduino : premiers pas en informatique embarqué**

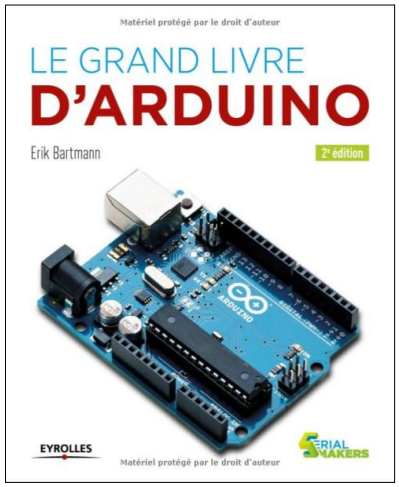
Ebook Arduino : eskimon.fr

GRATUIT



- **Le grand livre d'Arduino**

Environ 37€



Pour l'entraînement : toutes les fiches disponibles sur www.fonteniaud.fr/iut

ATTENTION : Aucune correction sur le site

VII. Résumé

Les parties indiquées ainsi sont à remplacer par les valeurs correspondantes à votre programme.
Broche : à remplacer SOIT par le numéro de la broche SOIT par le nom donné à la broche avant le setup.
Attention : Points virgules à la fin de chaque instruction. Ne pas oublier les parenthèses et les accolades. Utiliser uniquement les caractères alphanumériques.

2 slashs avant	`// cette ligne est un commentaire sur UNE SEULE ligne`	Commentaire sur **UNE SEULE ligne**
1 slash + 1 astérisque avant ET 1 astérisque + 1 slash après	`/*cette ligne est un commentaire sur PLUSIEURS lignes qui sera ignorée par le programme, mais pas par celui qui lit le code */`	Commentaire sur **PLUSIEURS lignes**

Avant le setup :

const int *nom_variable*=*broche* ;	Donner un nom à une broche
int *nom_variable*=*valeur* ;	Créer une variable 16 bits de type entier et lui donner une valeur initiale
long *nom_variable*=*valeur* ;	Créer une variable 32 bits de type entier et lui donner une valeur initiale
char *nom_variable*=*valeur* ;	Créer une variable 8 bits de type entier et lui donner une valeur initiale
float *nom_variable*=*valeur* ;	Créer une variable 32 bits de type entier et lui donner une valeur initiale
boolean *nom_variable* =*valeur* ;	Créer une variable booléenne et lui donner une valeur initiale

setup() :

pinMode(*broche*, OUTPUT) ;	Broche utilisée en sortie (actionneur)
pinMode(*broche*, INPUT) ;	Broche utilisée en entrée (capteur)

loop() (et setup()) :

nom_variable=digitalRead(*broche*) ;	**Lire capteur numérique**
nom_variable=analogRead(*broche*) ;	**Lire capteur analogique**

digitalWrite(*broche*, HIGH) ;	Mettre la sortie à l'état **haut**
digitalWrite(*broche*, LOW) ;	Mettre la sortie à l'état **bas**

delay (*durée en millisecondes*) ;	**Pause** en millisecondes (= 10^{-3} secondes)
delayMicroseconds (*durée en microsecondes*) ;	**Pause** en microsecondes (= 10^{-6} secondes)

while (*condition à valider pour exécution de la boucle jusqu'à ce qu'elle devienne fausse*) {
instructions à exécuter si la condition est vraie avant de l'exécuter ;
}
	Boucle while

do { *instructions à exécuter* ; } while(*condition à tester après chaque exécution de la boucle pour la relancer ou non*);	**Boucle do...while**

for (int *variable* ; *condition à tester avec la variable* ; *incrémentation de la variable*) {
instructions à exécuter tant que la condition est vraie ;
}
	Boucle for

while (1) {
instructions à exécuter ;
}
	Boncle infinie

void *NomLEDaFonction* (char *nom_variable*) {
instructions à exécuter ;
}
	Créer une fonction avec/sans paramètre, sans valeur de retour

NomLEDaFonction (*Valeur du ou des paramètre(s) à intégrer pour cette lecture*);	**Appeler une fonction avec/sans paramètre,** sans valeur de retour

Conditions :

if (*condition à tester*) {
instructions à exécuter si la condition est vraie ;
}
	if (si)

if (*condition à tester*) {
instructions à exécuter si la condition est vraie ;
}
else if (*condition à tester*) {
«instructions à exécuter si la condition est vraie» ;
}
	else if (sinon si)

if (*condition à tester*) {
instructions à exécuter si la condition est vraie» ;
}
else {
instructions à exécuter si la condition est fausse ;
}
	else (sinon)

Signes opératoires

Symboles	Signification
==	...est égale à...
<	...est inférieur à...
<=	...est inférieur ou égale à...
>	...est supérieur à...
>=	...est supérieur ou égale à...
!=	...est différent de...

Opérateurs logiques :

variable && *variable*	...et... (=and)
variable || *variable*	...ou... (=or)
!*variable*	inverse de... (=not)

Opérations :

variable + *variable*	Additions
variable - *variable*	Soustractions
variable * *variable*	Multiplications
variable / *variable*	Divisions
variable++	Incrémentation
variable--	Décrémentation

Bluetooth :

#include <SoftwareSerial.h>	Inclure la bibliothèque « SoftwareSerial » *
SoftwareSerial bluetooth (Rx, Tx) ;	Nouveau port série émulé (exmple : bluetooth) avec ports Rx et Tx (remplacer Rx et Tx par numéro de broche) *
bluetooth.begin(9600) ;	Initialisation port « bluetooth » (vitesse standard : 9600 bauds) **

if (bluetooth.available()) {
variable = bluetooth.read() ;
}
	Lorsque le module bluetooth donne des informations, les enregistrer dans une *variable* **

* Ne pas utiliser ces instructions si le module Bluetooth est branché sur les broches Rx et Tx natives de l'Arduino (0 et 1)
** Si le module Bluetooth est branché sur les broches Rx et Tx natives d'un Léonardo remplacer bluetooth par Serial1

Port série (ordinateur) :

Serial.begin(9600) ;	Initialisation port série (vitesse standard : 9600)
Serial.print("texte à envoyer") ;	Message **sans** saut de ligne
Serial.println("texte à envoyer") ;	Message **avec** saut de ligne
Serial.print(*variable*) ;	Valeur variable **sans** saut de ligne
Serial.println(*variable*) ;	Valeur variable **avec** saut de ligne