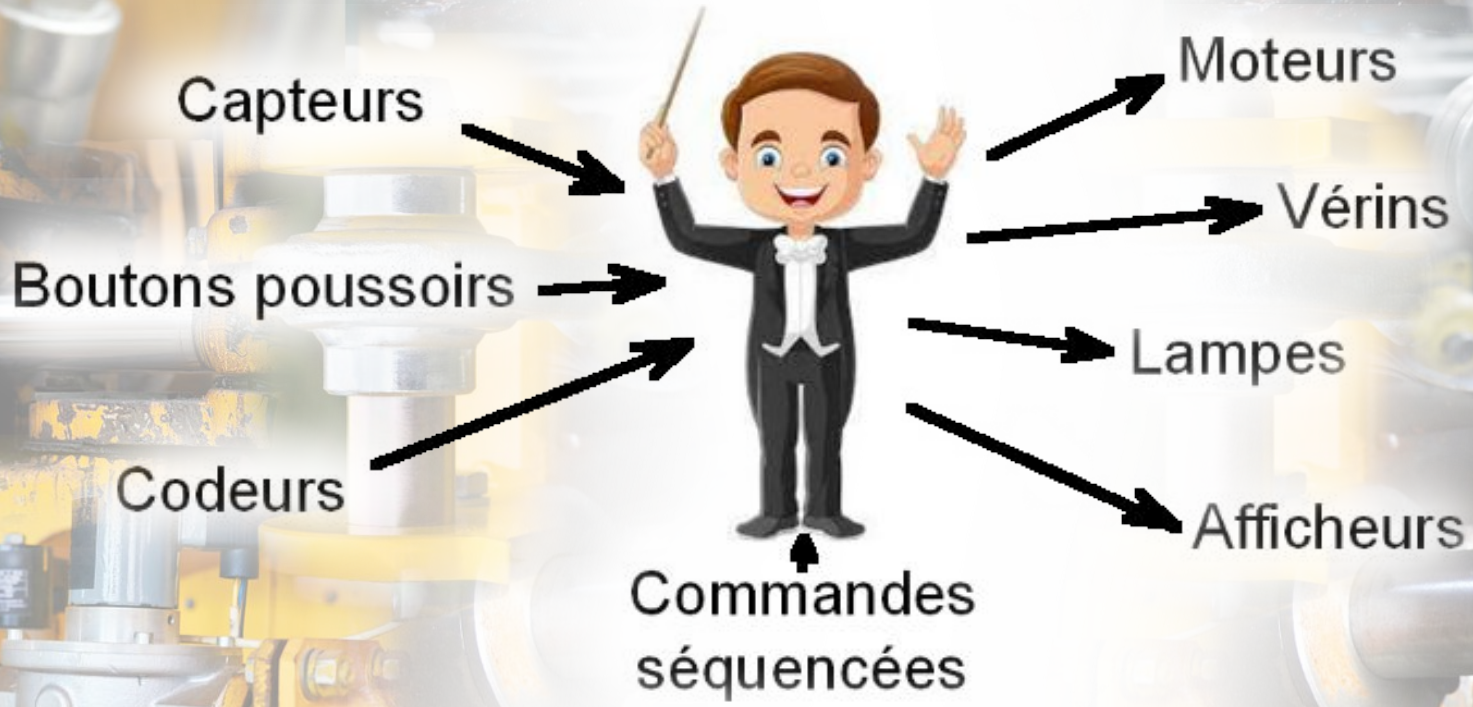
IDROLIK
AUTOMATISATION HYDRAULIQUE

Arduino - premier contact



IDROLIK
AUTOMATISATION HYDRAULIQUE

Sommaire

IDROLIK
AUTOMATISATION HYDRAULIQUE

Information préliminaire

- Cette présentation est accessible via le site idrolik.com
  - rubrique documentation
- Elle sera mise à jour sur le site au fur et à mesure qu'elle évoluera
- De cette manière, vous aurez accès aux exemples de programmes pour vérifier que ça marche

IDROLIK
AUTOMATISATION HYDRAULIQUE

Introduction

- Aujourd'hui on a tous envie d'automatiser certaines tâches
  - Domotique, alimentation du chat en absence, cibles pour le tir au pistolet, poêle à granulés, machine à café, jeux de lumière ...
- Deux principales classes de circuits électroniques sont conçus pour faire tourner des programmes informatiques
  - Les microprocesseurs et les microcontrôleurs
- Le microprocesseur est taillé pour réaliser des traitements lourds
  - En termes de quantité de calculs
  - En termes de quantité de mémoire nécessaire
  - En termes de volumes de données manipulées (disques, cartes graphiques, réseaux, ...)
  - En termes de qualité d'interface homme-machine (affichage, souris, pad ...)
- Le discret microcontrôleur est taillé pour une action efficace au plus près du matériel

Vocabulaire – bit & octet

- **Un bit est l'information binaire minimale**
  - Est égal à 0 ou à 1
  - En électronique
    - 0 correspond souvent à une tension électrique nulle (ou proche de zéro volt)
    - 1 correspond souvent à une tension de 5 volts (ou proche de 5 volts)
- **Un octet est un ensemble ordonné de 8 bits**
  - C'est la quantité de base pour la mémorisation et le traitement d'informations numériques
  - On parle ...
    de kilo octets (kilo = 10^3),
    de méga octets (mega = 10^6),
    de giga octets (giga = 10^9)
    de tera octets (tera = 10^{12})
- **Dans un octet, on parle de …**
  - Bit de poids faible (bit de droite)
  - Bit de poids fort (bit de gauche)
  - Un octet représente un nombre écrit en base 2 (deux signes 0 et 1)
    Il peut prendre n'importe quelle valeur entière entre 0 et 255
    $01101010_{base\ 2} = 2^1 + 2^3 + 2^5 + 2^6 = 106$

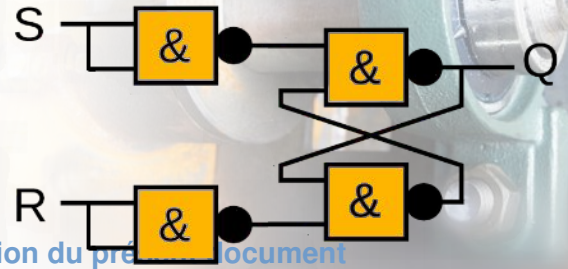
Vocabulaire – mémoire

- Imaginez que vous posiez 8 verres alignés sur une table, tantôt couchés, tantôt debout
  - En faisant ça, vous venez de créer une mémoire d'un octet
- Vous partez et vous revenez un an après
  - Si personne n'a modifié l'état de votre mémoire, vous retrouvez votre octet comme vous l'avez laissé
  - Dit autrement, votre octet est resté en mémoire
- En électronique, on parle de :
  - ROM : Read Only Memory (pour mémoriser un octet en ROM, on peut imaginer 8 fils à 5 volts et dont certains sont coupés et mis à la masse pour faire des zéros)
  - RAM : Random Access Memory (pour mémoriser un octet en mémoire vive, on peut imaginer commander 8 bascules électroniques dites « RS »)
- Pour un microcontrôleur, on parle de kilo octets

Bascule RS :
S et R à zéro
S passe à 1, Q passe à 1
S repasse à zéro, Q reste à 1
R passe à 1, Q passe à zéro
R repasse à zéro, Q reste à zéro

Cellule « & » :
2 entrées à 1 ► sortie à 1
Sinon, sortie à zéro

« ● » :
Inversion de la sortie



Vocabulaire – liaison série

- **Principe de la liaison série**
  - Un circuit électronique émetteur impose le niveau de tension (0 ou 1) sur un fil à chaque top d'horloge
  - On parle de fréquence d'horloge
  - Le circuit récepteur lit à la même fréquence le niveau de tension sur le fil
  - Les circuits émetteur et récepteur peuvent avoir chacun leur horloge (oscillateur à quartz) qu'ils règlent à la même fréquence
  - Ou la fréquence d'écriture des bits sur la ligne peut être partagée par l'émetteur sur un fil du bus
- **Pour une liaison radio, c'est pareil …**
  - Les niveaux de tension sur un fil deviennent des trains d'ondes radio émis en série dans l'air
- **Lorsqu'on veut envoyer des informations complexes, on constitue des trames**
  - Une trame est souvent constituée par un grand nombre d'octets
  - Chaque trame commence par une en-tête qui indique (entre autres) l'adresse du destinataire, et celle de l'émetteur (ce qui permet au destinataire de répondre)
  - Chaque bit de chaque octet de la trame est « émis en série »

Note : il y a 30 ans, les communications série fonctionnaient à une fréquence de 9600 bauds (= environ 10000 changements d'état / seconde). Aujourd'hui, Ethernet est en Gigabit (plusieurs millions de changements d'état / seconde) Donc l'image devrait représenter un TGV …

Vocabulaire – le bus

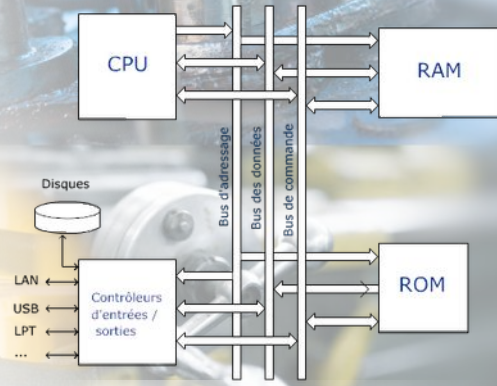
- **Imaginons des prisonniers, chacun dans sa cellule ...**
  - Ils ne peuvent pas se voir, ils ne peuvent pas s'entendre
  - Les cellules sont reliées entre elles par un réseau de radiateurs connectés par des tuyauteries métalliques
  - Chacun peut taper sur son radiateur pour émettre des messages par exemple avec le code morse

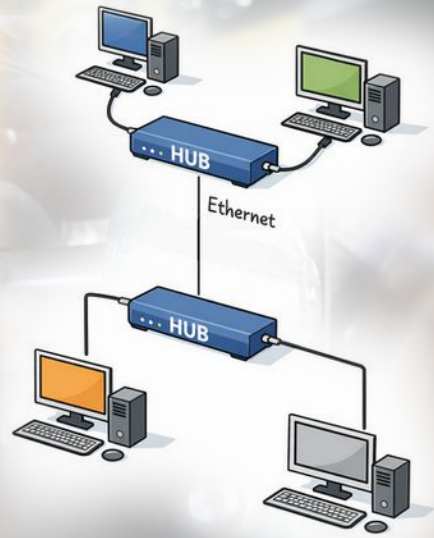
  - Tous les prisonniers entendent lorsque l'un d'entre eux frappe sur son radiateur
  - Si un prisonnier veut communiquer avec un autre, il fait précéder son message du numéro de la cellule du destinataire et de son propre numéro de cellule.
    Seul le destinataire prend le message pour lui, les autres l'ignorent (si on veut de la discrétion, le corps des messages peut être chiffré)

- **Analogie : ici, le réseau de radiateurs constitue un bus de données, toutes les informations transitent par ce canal**
  - Arbitrage à la mode Ethernet :
    - Règle 1 : si quelqu'un est en train de parler, les autres se taisent
    - Règle 2 : si deux personnes prennent la parole en même temps, le message est incompréhensible. Tout le monde se tait et initialise un temps d'attente aléatoire avant de reprendre la parole ... et la règle 1 s'applique
  - Arbitrage à la mode USB : le maître de cérémonie distribue la parole, les autres ne parlent que lorsqu'ils y sont invités

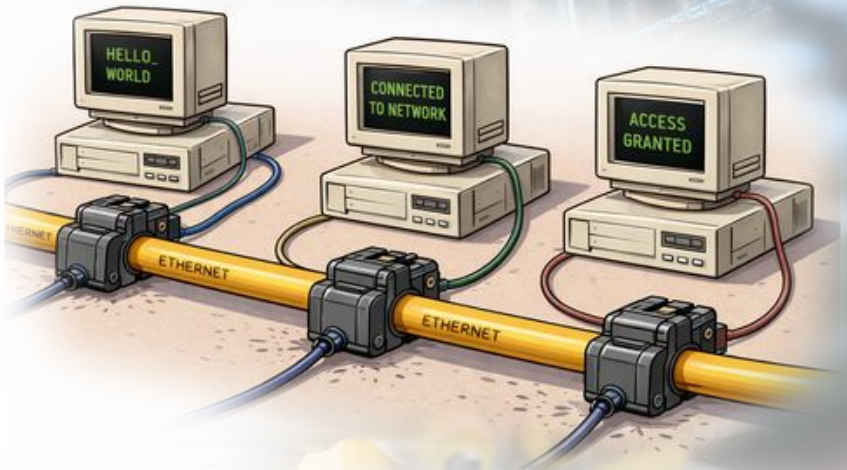
Vocabulaire - Le bus

- Dans la suite, le mot « bus » est souvent utilisé
- En informatique, un bus ...
  - Est une liaison sur laquelle on peut connecter plusieurs circuits électroniques
    - Sur une carte électronique, le bus peut prendre la forme de plusieurs conducteurs électriques en parallèle sur lesquels sont directement connectés les interfaces des périphériques.
      - Si un circuit « parle » sur le bus (= impose les niveaux de tensions sur les conducteurs du bus), les autres circuits « se taisent » (= se contentent de « lire » les niveaux de tension)
      - Les périphériques ne prennent en compte que ce qui leur est adressé
  - Elle peut être constituée des liaisons point-à-point reliées en étoile(s) par des nœuds actifs (hubs ou switches)
    - Un format spécifique de connecteurs matériels est souvent prévu pour éviter qu'un circuit non compatible puisse être connecté et brouille les communications sur le bus.
  - Est un « lieu » où tous les dispositifs électroniques connectés peuvent être destinataires de messages
  - Est associé à un protocole de communication dédié
    - Le protocole définit et organise les échanges, avec un système d'adressage qui permet d'indiquer qui parle et qui est destinataire
- Par opposition au bus, on parle de liaisons point à point.
  Avec une liaison point-à-point, pas besoin de dire à qui s'adresse le message

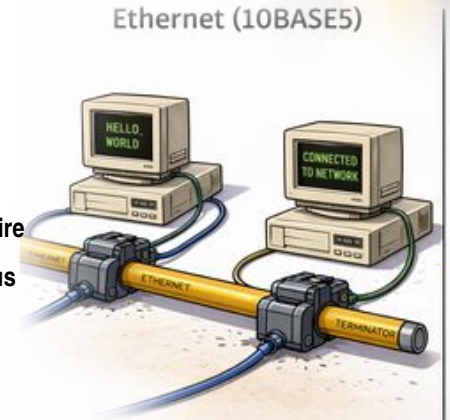




Exemple 1 – ETHERNET



- **A l'origine Ethernet était un protocole qui utilisait un câble coaxial unique**
  - Toutes les machines étaient connectées dessus avec des prises vampires
  - Adresses Ethernet uniques sur 48 bits (6 octets)
  - Une adresse « broadcast » permettait à une machine de demander à la cantonade qui fournit tel ou tel service.
    Exemples « qui peut me fournir une adresse IP ? », ou « y-a-t-il des imprimantes réseau ? » ...
  - Protocole basé sur des trames avec en-tête indiquant (entre autres) les adresses source et destination
  - Les machines émettaient leurs trames en série sur le câble coaxial
  - Lorsque deux machines émettaient en même temps, détection de la collision par l'ensemble des machines
    - Les machines stoppaient leur émission et initialisaient une temporisation aléatoire
    - La première machine qui émettait prenait le bus, les autres attendaient que le bus se libère
- **Le câble coaxial constituait un bus**
  - Par certains cotés, le WIFI (qui a son protocole spécifique) ressemble à Ethernet coaxial

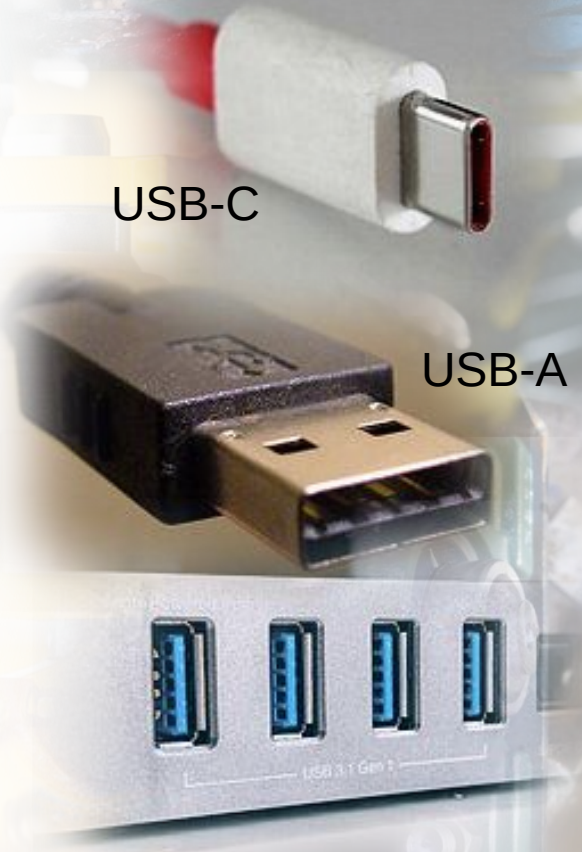


Exemple 1 - ETHERNET

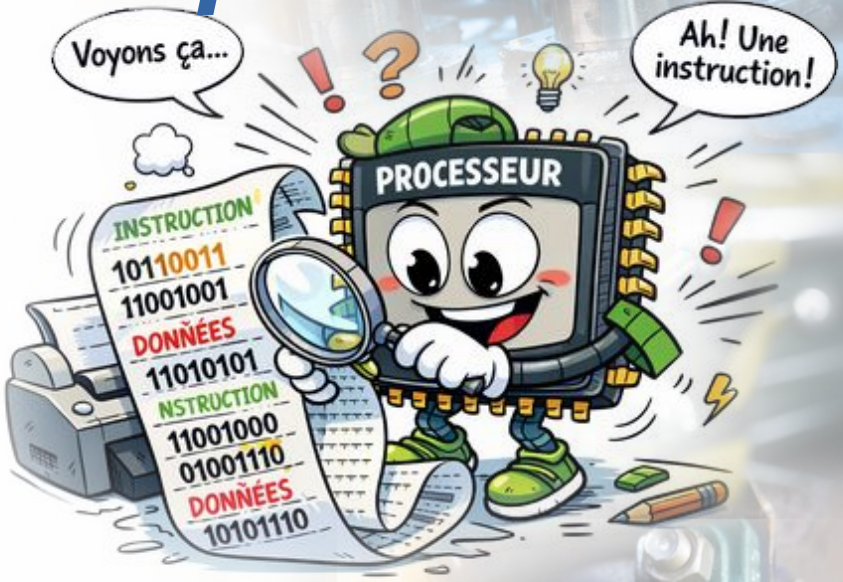
- **Aujourd'hui, Ethernet s'est modernisé**
  - 4 paires de fils « full duplex » (transmission simultanée dans les deux sens sur chaque paire)
- **Propagation des trames via des « switches » qui ...**
  - transmettent les trames reçues sur un port vers les autres ports
  - apprennent de quel port arrivent les trames de chaque adresse Ethernet afin de limiter le trafic
- **Envoi toujours possible de trames « broadcast » (= à destination de tous les dispositifs présents sur le réseau local)**
  - Recherche serveur(s) DHCP (Dynamic Host Configuration Protocol)
  - Recherche d'imprimante(s), de scanner(s) ...

Exemple 2 - USB

- Clavier, souris, disque dur, clé USB (Universal Serial Bus) ...
- 4 conducteurs (hors alim du périphérique)
  - 2 conducteurs D+ et D- pour transmettre les données en mode série (un sens à la fois)
  - 2 conducteur VCC et GND (tension alimentation et masse)
- Adresse USB sur 7 bits (127 périphériques maxi sur un bus USB)
- Protocole basé sur des liaisons série suivant le principe maître-esclave (1 maître, plusieurs esclaves)
  - Un esclave ne parle jamais spontanément, même pour se présenter.
    Sa présence est détectée à la connexion par une résistance pull-up sur une ligne de donnée
  - Par exemple, lorsque l'équipement USB est un clavier, l'ordinateur lui fait régulièrement appel pour savoir si une touche a été pressée (polling)
- Les hubs USB (si présents) sont interrogés périodiquement par le maître pour connaître les éventuels nouveaux périphériques connectés derrière



Vocabulaire : processeur



- Un processeur est un composant construit pour
  - Accéder à des circuits de mémoire informatique via un bus d'adresses et un bus de données
  - Lire des octets en mémoire et les placer dans des registres internes
  - Interpréter certains octets comme des instructions et effectuer des opérations entre registres sur les données
  - Écrire les résultats du traitement en mémoire
- Le processeur possède un registre interne particulier appelé pointeur d'instruction.
  - Ce registre contient l'adresse mémoire de la prochaine instruction à lire
- Les opérations sont cadencées par les tops d'une horloge interne (oscillateur)
- Certaines adresses mémoire correspondent en réalité à des registres de circuits d'entrées-sorties
  - Ils communiquent avec des périphériques via des bus
    → Contrôleur de disque dur, carte graphique, contrôleur USB ...

IDROLIK
AUTOMATISATION HYDRAULIQUE

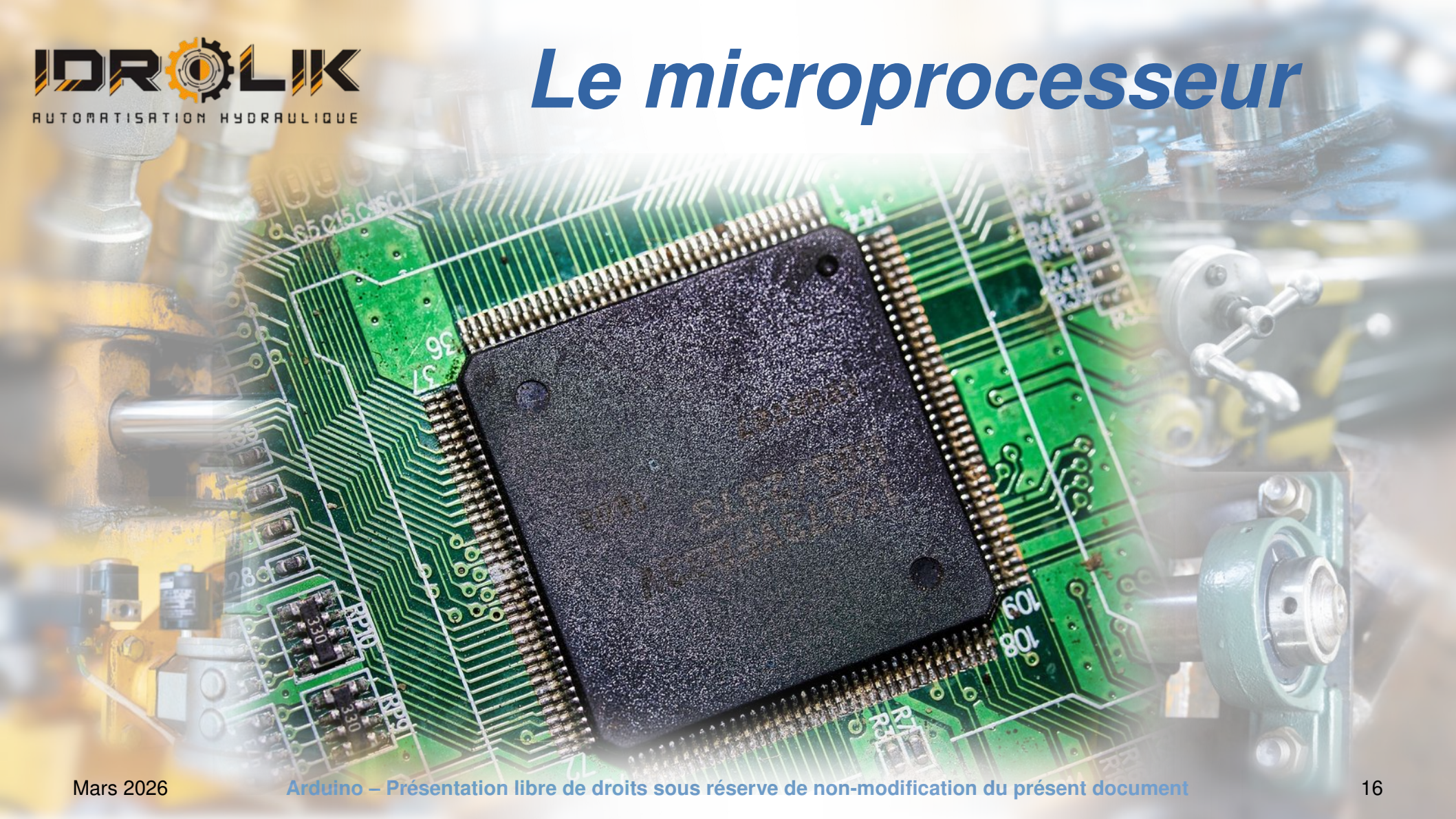
1 - Microprocesseur versus microcontrôleur

Le microprocesseur

- Est plus complexe (nombre de transistors et structure interne) et plus performant que le microcontrôleur
  - Rapidité, capacité de calcul (coprocesseur arithmétique intégré, multi cœur), quantité de mémoire vive adressable
- Chauffe davantage, il a besoin d'un système de refroidissement (ventilateur et/ou circuit d'eau)
- Est l'élément central d'une carte mère, il s'interface matériellement :
  - Avec des circuits de mémoire via des bus d'adresse et de donnée
  - Avec des périphériques via des bus d'entrées-sorties
    - Carte graphique, disques durs, clavier, souris, Ethernet, Wifi...
- Fonctionne avec :
  - Un BIOS (Basic Input Output System) stocké dans le microprocesseur lui-même (firmware)
    - permet l'utilisation du disque dur afin de charger le système d'exploitation dans la mémoire du microprocesseur (bus SATA ...)
  - Un système d'exploitation qui souvent stocké sur un stockage de masse (disque dur...) et qui prend en charge
    - Le fonctionnement en multitâche (en coopération avec les modes internes du microprocesseur : mode utilisateur, mode noyau ...)
    - La gestion des périphériques (clavier, souris, imprimantes, écrans, scanners …)
    - Il fournit les interfaces systèmes pour les logiciels applicatifs (affichage, Internet, ...)

IDROLIK
AUTOMATISATION HYDRAULIQUE

Le microprocesseur

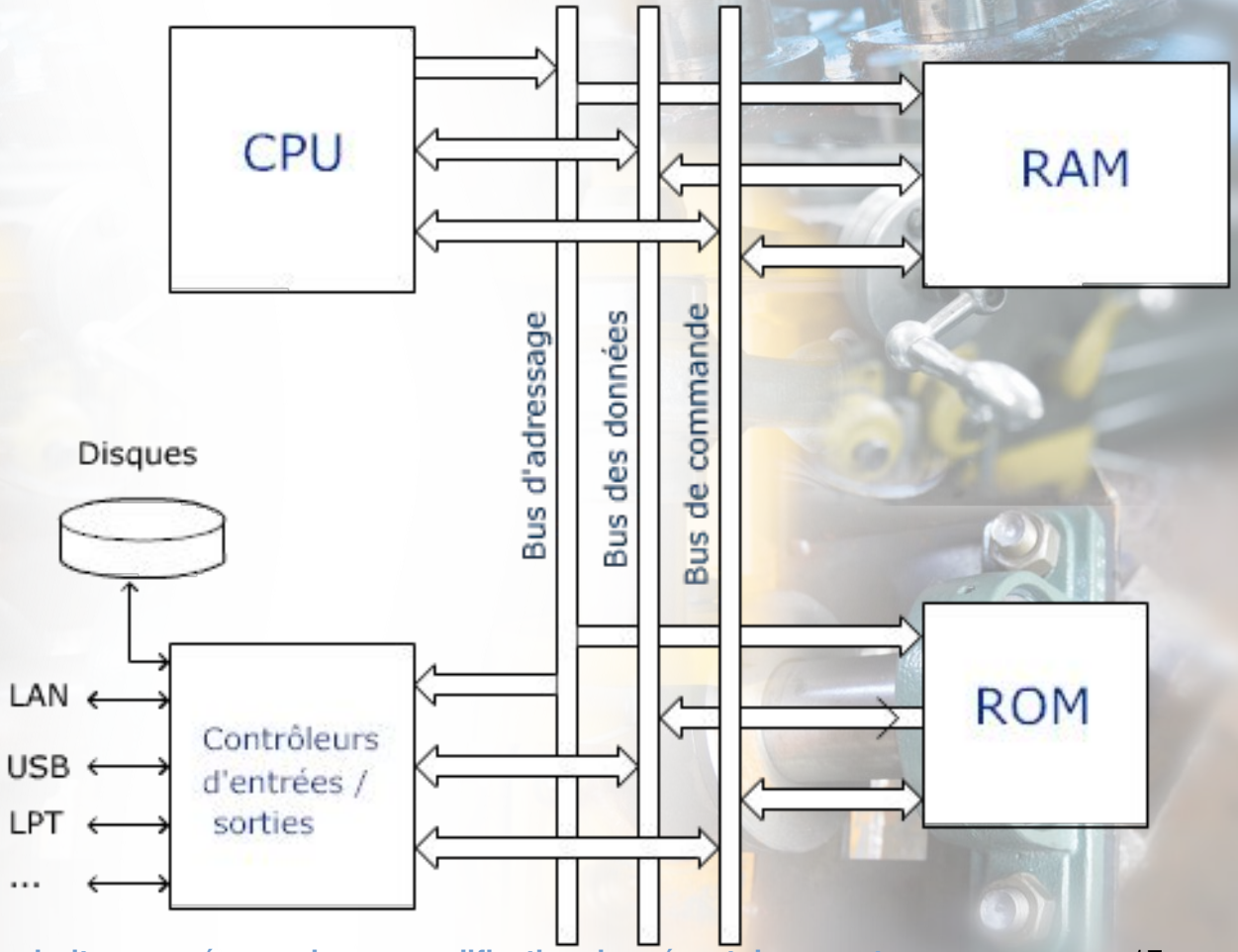


µprocesseur / carte mère

CPU : Central Processing Unit (**Microprocesseur**)
RAM : Random-Access Memory (Mémoire vive)
ROM : Read only memory
LAN : Local area network (typiquement : Ethernet)
USB : Universal Serial Bus
LPT : line printing terminal

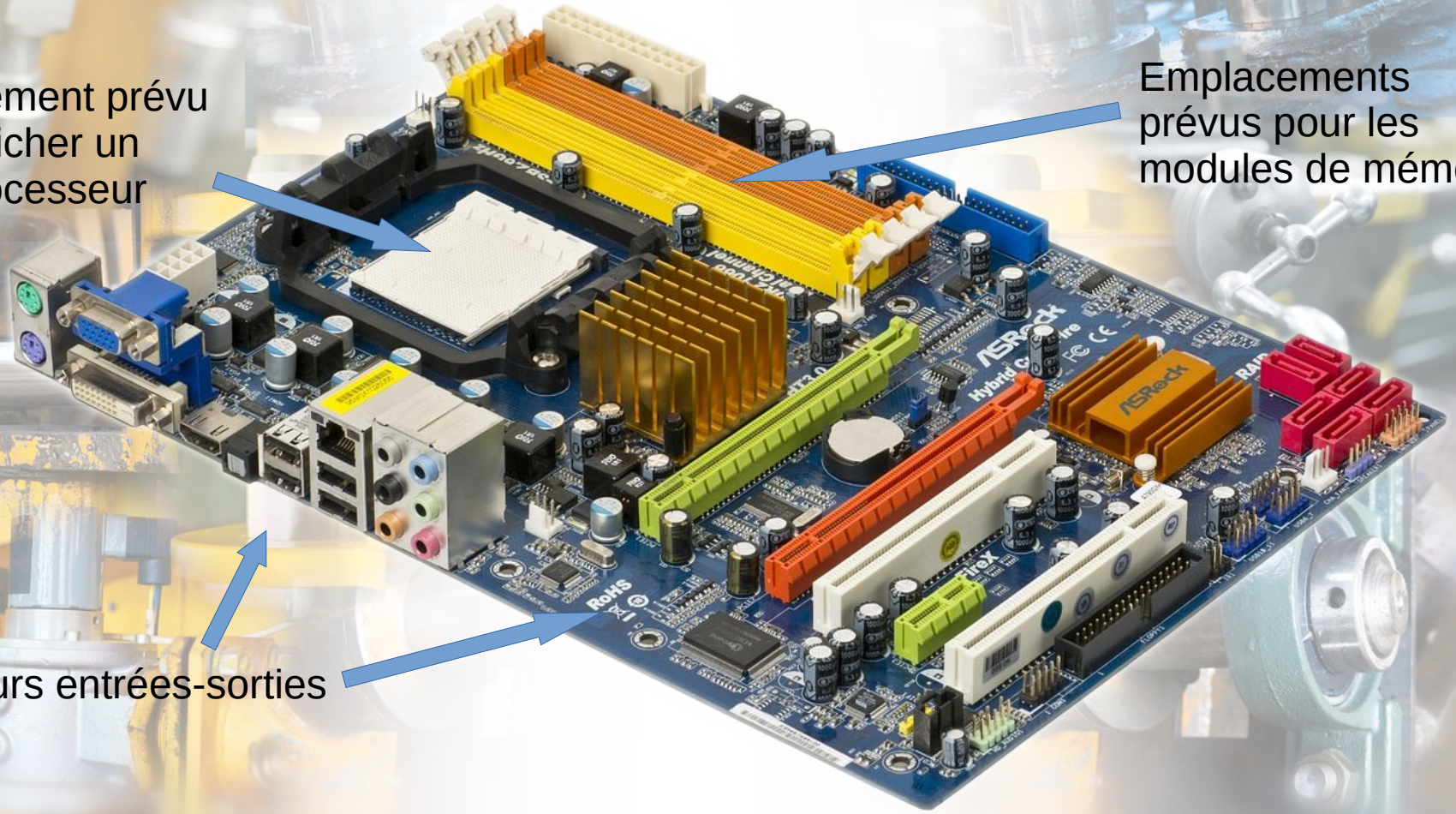
Bus d'adresse : 32 conducteurs si adressage sur 32 bits

Bus de données : 16 conducteurs si les circuits manipulent des mots de 16 bits



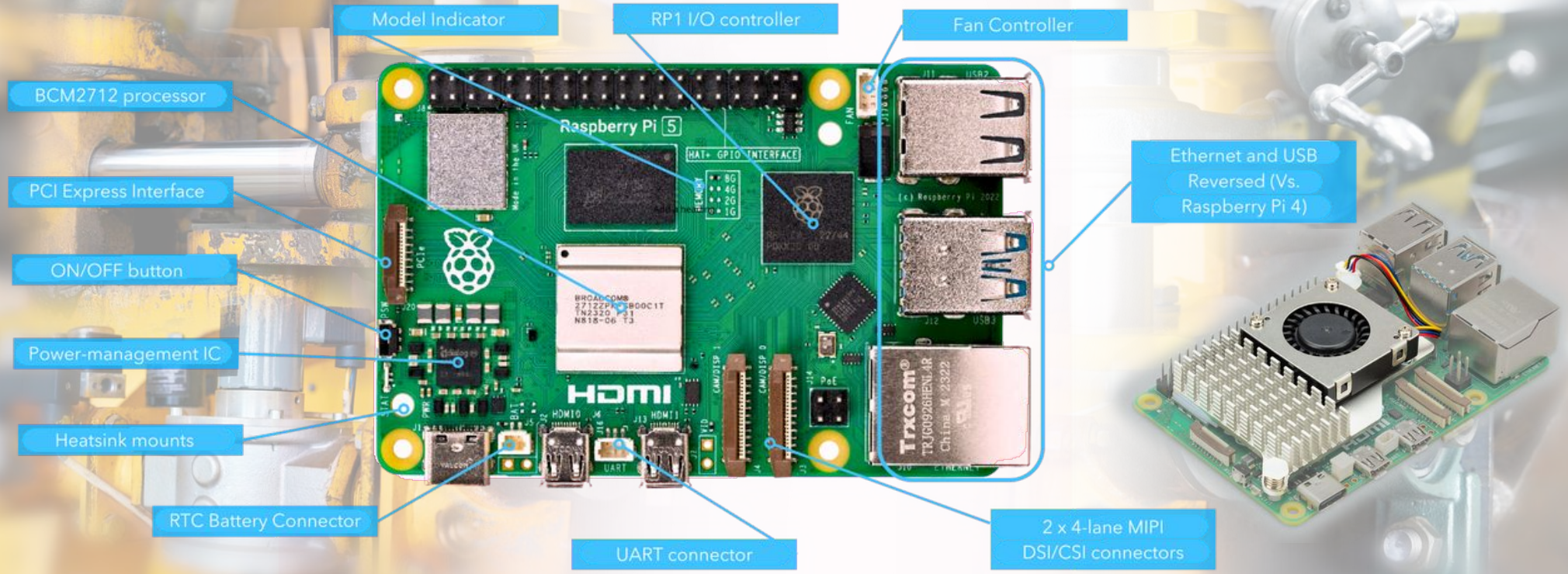
IDROLIK
AUTOMATISATION HYDRAULIQUE

Carte mère



Raspberry Py

- **Un ordinateur qui ressemble à une carte électronique**
  - **Mémoire vive jusqu'à 8 Go, carte micro SD jusqu'à 128 Go qui sert de disque dur, pour héberger système d'exploitation et fichiers**
  - **Tout ce qu'il faut pour connecter clavier et écran, communiquer et faire tourner des programmes applicatifs**

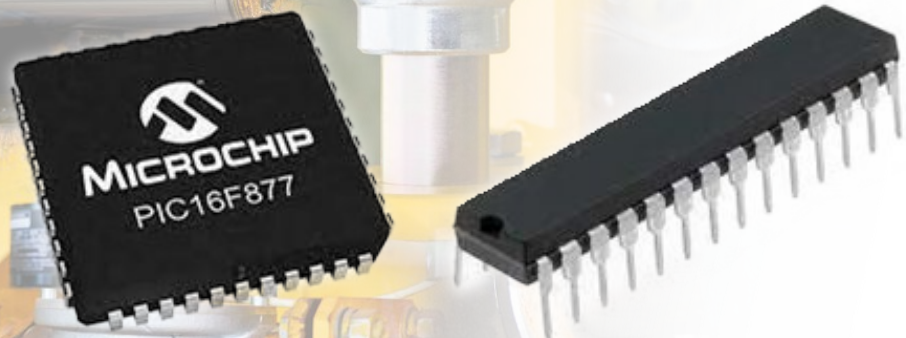


Raspberry et Arduino

- Raspberry et Arduino : collaboration possible
- Raspberry Pi (le “cerveau”)
  - Mini-ordinateur qui exécute un système Linux
  - Gère l’écran, le clavier, le Wi-Fi, Internet
  - Exécute des programmes applicatifs
  - Stocke des données et propose une interface (écran, web)
- Arduino (les “réflexes”)
  - Microcontrôleur dédié au pilotage matériel
  - Lit des capteurs (température, lumière, bouton…)
  - Commande des actionneurs (LED, relais, moteurs, servos…)
  - Démarre instantanément et fonctionne en temps réel
- Échange d’informations :
  - USB, UART, I2C

Avec le microcontrôleur

- Pas de carte mère
- Mise en œuvre simplifiée

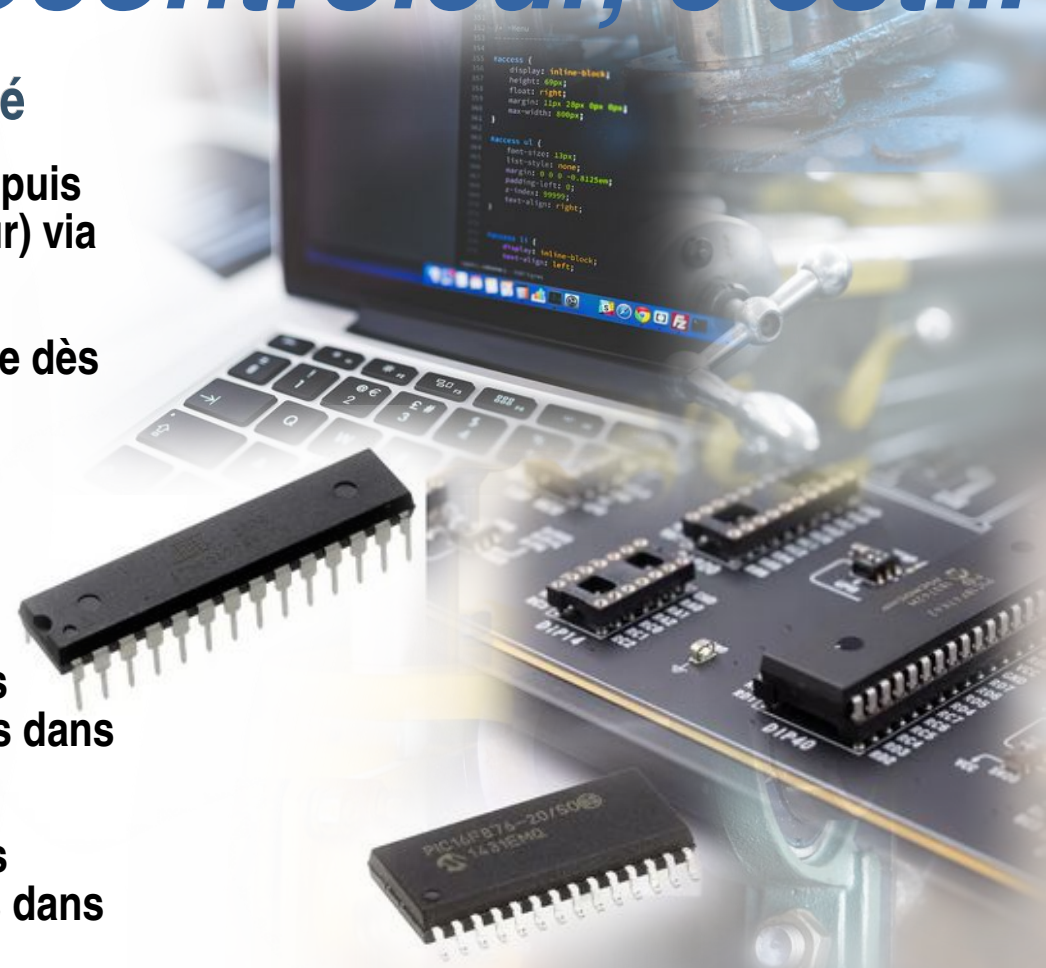


Le microcontrôleur

- Embarque tout ce qui est nécessaire à son fonctionnement dans un seul circuit
  - Processeur, mémoire, interfaces d'entrées-sorties
- Permet de séquencer automatiquement des opérations (programme informatique)
  - Avec des variables, des boucles, des tests, des fonctions…
- Permet de communiquer avec d'autres circuits électroniques
  - USB, liaison série, bus I2C, ...
- Permet de récupérer des informations sur l'environnement :
  - Boutons poussoirs, capteurs fin-de-course, capteurs de proximité (inductifs, capacitifs, ultrasons…), capteurs de température, de lumière, d'accélération, GPS…
- Permet de commander des organes de puissance (via un relais ou une électronique de puissance) :
  - Chauffage, lumière, sirène, moteur, vérin ...

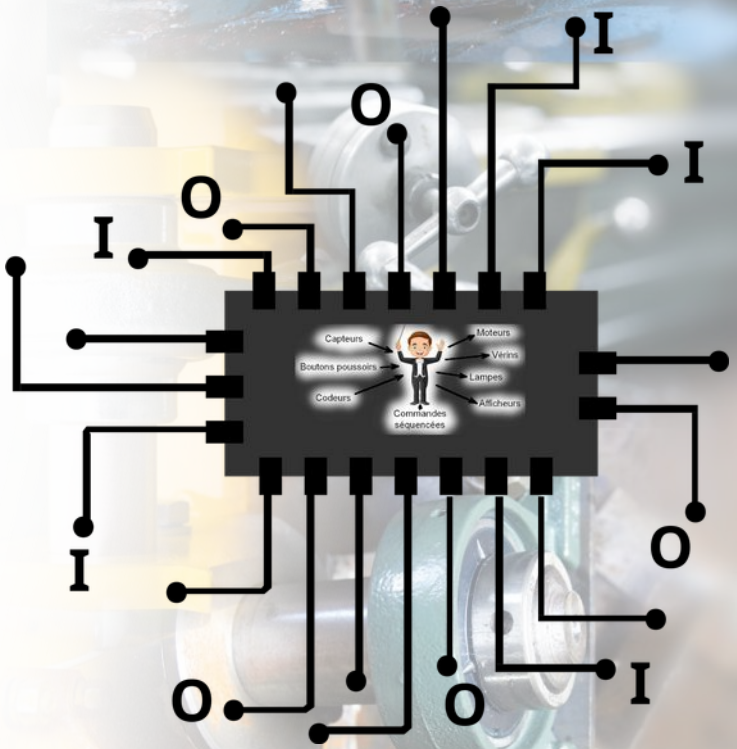
Un microcontrôleur, c'est...

- Un circuit électronique micro programmé
  - Le programme est préparé sur ordinateur puis téléversé (ordinateur vers microcontrôleur) via un câble USB
  - Le programme s'exécute ensuite en boucle dès la mise sous tension du microcontrôleur (fonctionnement en autonomie)
- Un circuit électronique qui présente des broches (contacts)
  - Certaines broches peuvent être désignées comme des entrées (à l'aide d'instructions dans le programme)
  - Certaines broches peuvent être désignées comme des sorties (à l'aide d'instructions dans le programme)



Un microcontrôleur, c'est...

- **Un circuit électronique dont les broches désignées comme des entrées peuvent être lues par le programme**
  - Une tension proche de 5 volts appliquée sur une broche d'entrée numérique est interprétée comme un 1
  - Une tension proche de 0 volt appliquée sur une broche d'entrée numérique est interprétée comme un 0
  - Certaines entrées peuvent être configurées pour lire une tension analogique entre 0 et 5V
  - Les entrées permettent de lire l'état de boutons poussoirs, de capteurs …
- **Un circuit électronique dont les tensions sur les broches désignées comme des sorties peuvent être mises à 0 ou à 1 par le programme**
  - Une sortie mise à 1 applique une tension de 5 volts sur la broche correspondante
  - une sortie mise à 0 applique une tension de 0 volt sur la broche correspondante
  - Les sorties permettent de commander des lampes, des moteurs …



En résumé

- **Avec le microcontrôleur :**
  - Pas de clavier/souris comme l'ordinateur, plutôt des boutons poussoirs, des potentiomètres, des boutons rotatifs, des claviers à code ...
  - Pas d'affichage graphique évolué (fenêtres avec gestion devant/derrière…), plutôt des LED, des afficheurs à une ou plusieurs lignes de caractères ...
  - Pas de programme applicatif lourd, plutôt des programmes courts et astucieux, le talent du programmeur s'exprime dans la concision et l'efficacité
  - Pas de multitâche massif géré par un système d'exploitation, plutôt une boucle de surveillance rapide et des traitements sur interruption ...
- **Ses forces :**
  - Le microcontrôleur est aisé à mettre en œuvre
  - Le microcontrôleur est adapté à la gestion d'automatismes au plus près du matériel
  - Le microcontrôleur peut communiquer en direct avec d'autres microcontrôleurs et avec le monde extérieur via des modules électroniques additionnels spécialisés

Exemple machine à café

- **Entrées microcontrôleur**
  - Bouton de sélection café long ou café court
  - Bouton poussoir de lancement d'un cycle
  - Capteur de niveau d'eau mini dans le réservoir d'eau
  - Sonde de température d'eau dans l'unité de chauffage
- **Sorties microcontrôleurs**
  - Résistance de chauffage de l'eau
  - Pompe pour pousser l'eau chaude sous pression à travers la dosette

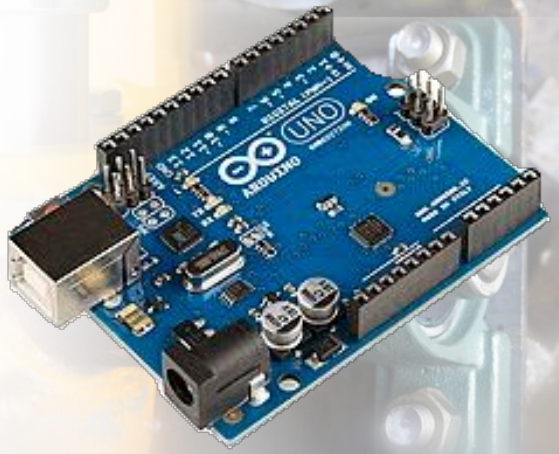
- **Boucle programme microcontrôleur**
  - Attendre l'appui sur le bouton « démarrage »
  - Activer (mettre à 1) la sortie « chauffage de l'eau »
  - Attendre que l'eau soit chaude (surveillance entrée sonde de température)
  - Activer (mettre à 1) la sortie « pompe »
  - Attendre X secondes suivant entrées « café long / café court », attente interrompue si bouton « démarrage » appuyé pendant le fonctionnement
  - Mettre à 0 la sortie « chauffage de l'eau »
  - Mettre à 0 la sortie « pompe »
  - Pendant le cycle, surveiller le capteur de niveau d'eau et mettre les sorties à 0 si manque d'eau détecté

IDROLIK
AUTOMATISATION HYDRAULIQUE

• 2 - Arduino, qu’est-ce que c’est ?

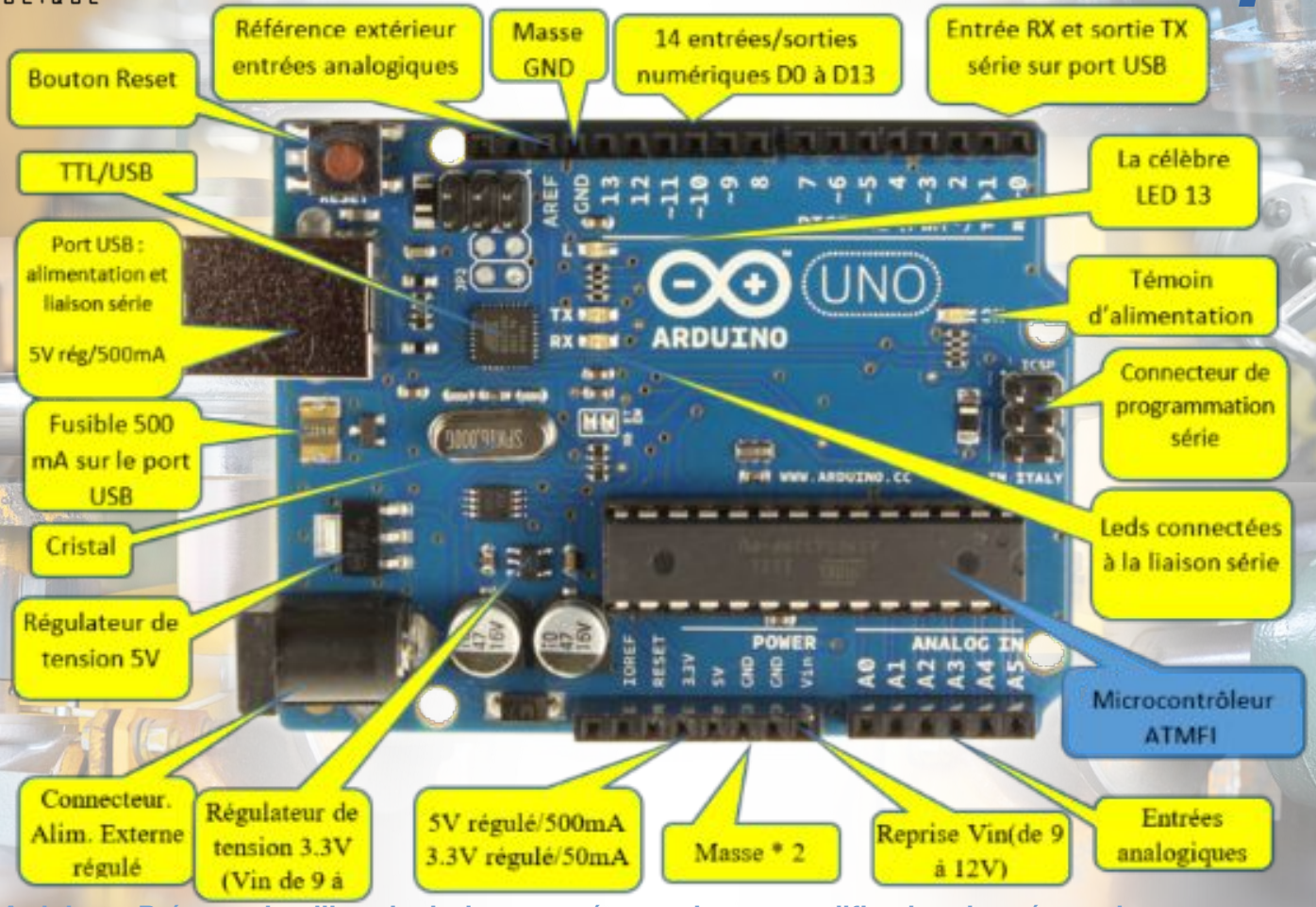
Arduino

- L'environnement Arduino intègre
  - Des modules électroniques basés sur des microcontrôleurs (Atmega …)
  - Chaque module est équipé du petit nécessaire pour fonctionner (oscillateur, connecteurs ...)
  - Une interface logicielle de développement disponible sur Linux et Microsoft Windows
- Qu'est-ce qu'Arduino a de particulier?
  - Arduino est largement répandu
  - C'est à la fois une société et une vaste communauté open source
  - Le matériel n'est pas cher
  - L'ensemble est très bien documenté, avec une communauté importante
  - L'interface de programmation et de téléversement est open source, pratique et gratuite



IDROLIK
AUTOMATISATION HYDRAULIQUE

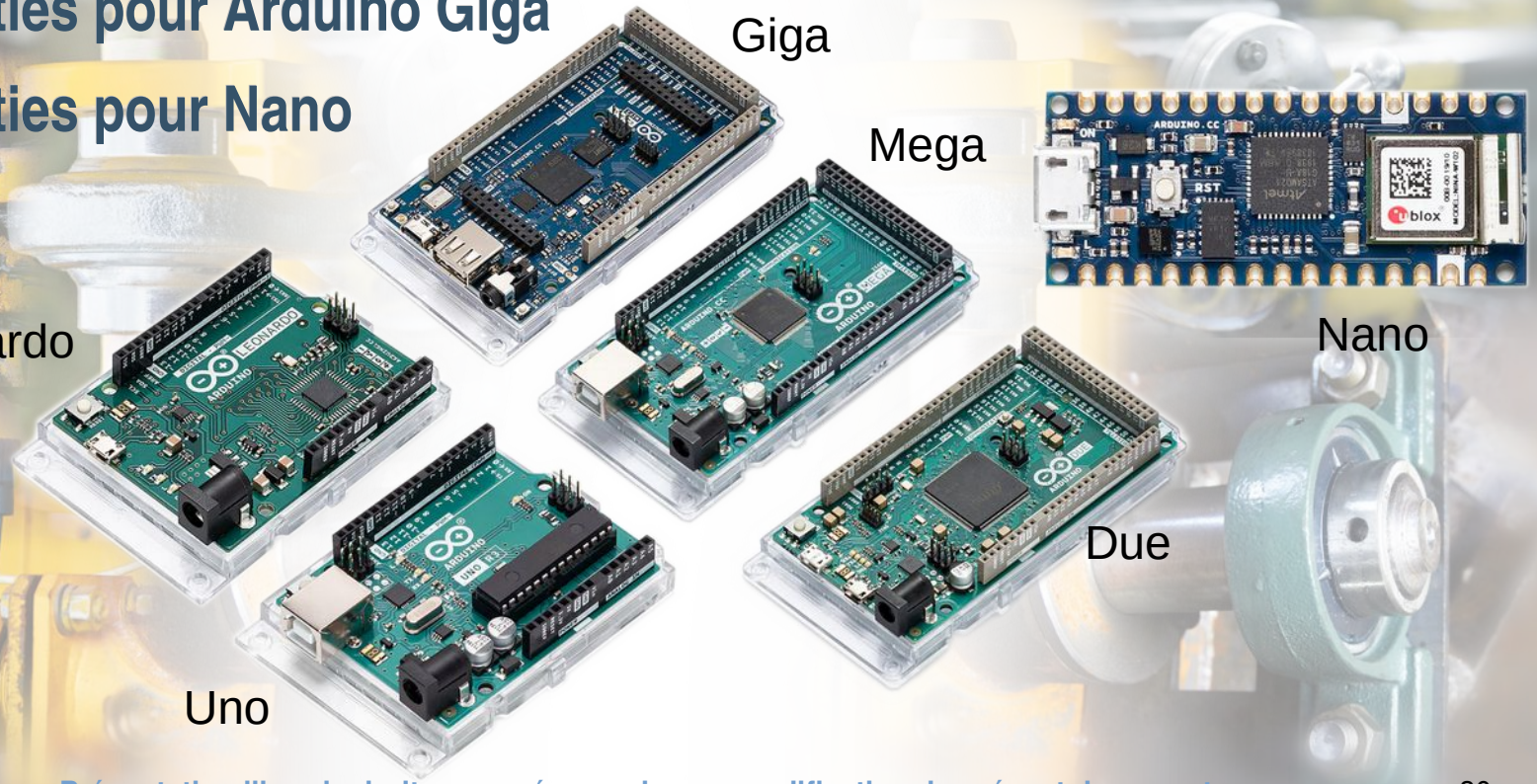
Module : on trouve quoi ?



IDROLIK
AUTOMATISATION HYDRAULIQUE

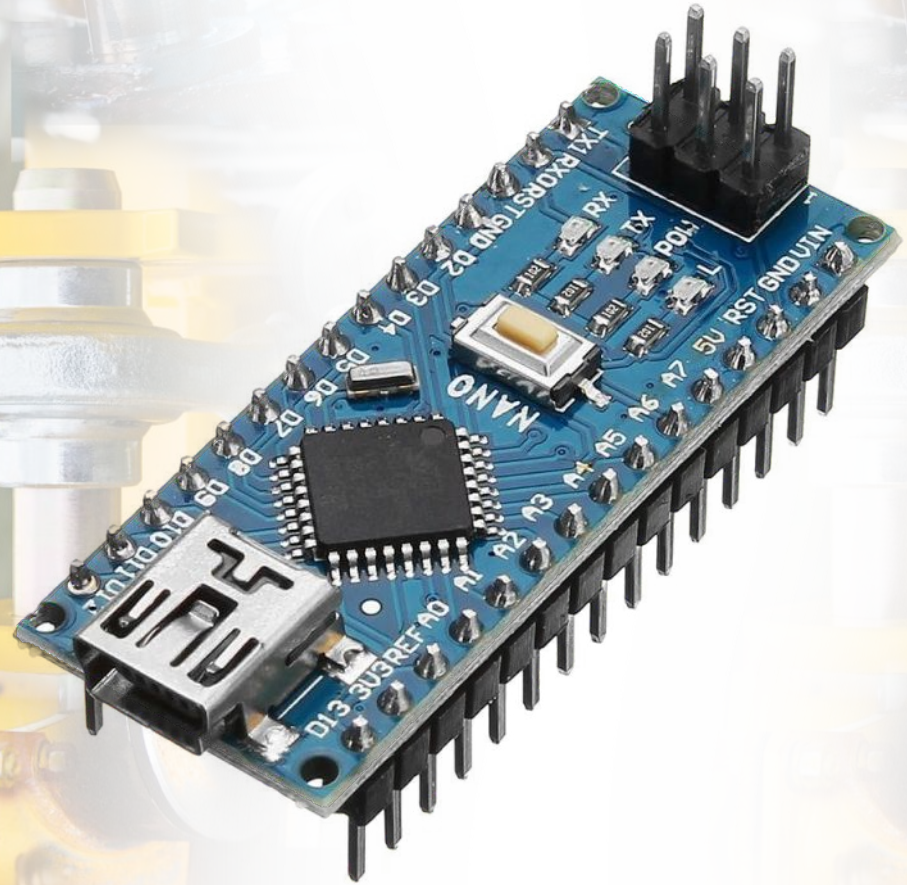
Arduino

- Propose plusieurs modules basés sur différents microcontrôleurs
- 76 entrées-sorties pour Arduino Giga
- 14 entrées-sorties pour Nano



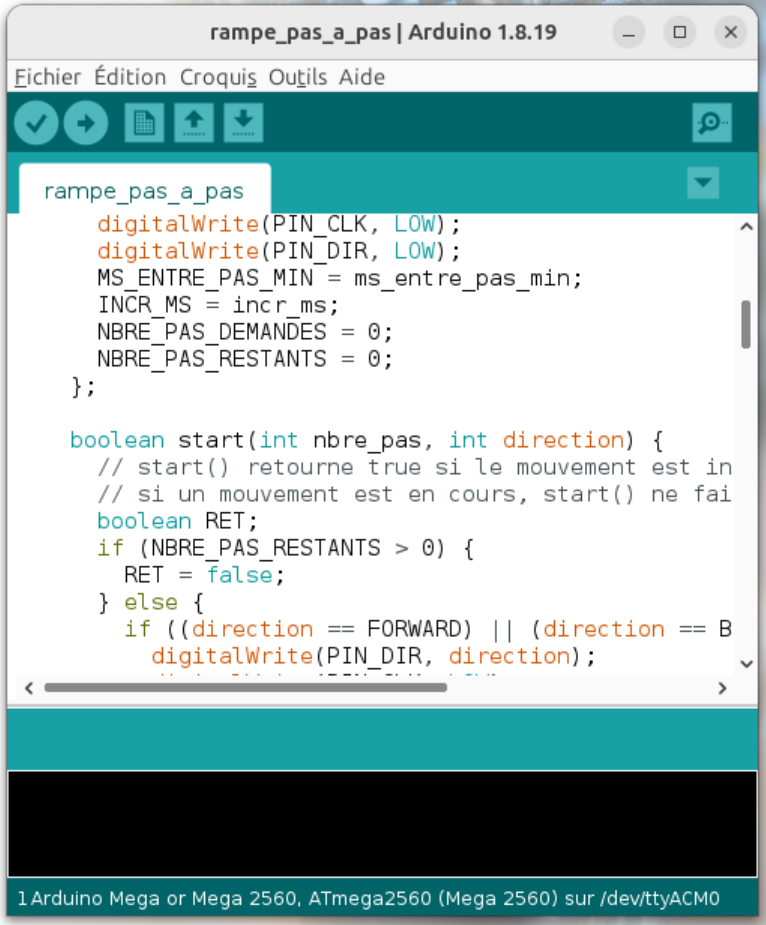
IDROLIK
AUTOMATISATION HYDRAULIQUE

Le petit de la série



IDE Arduino

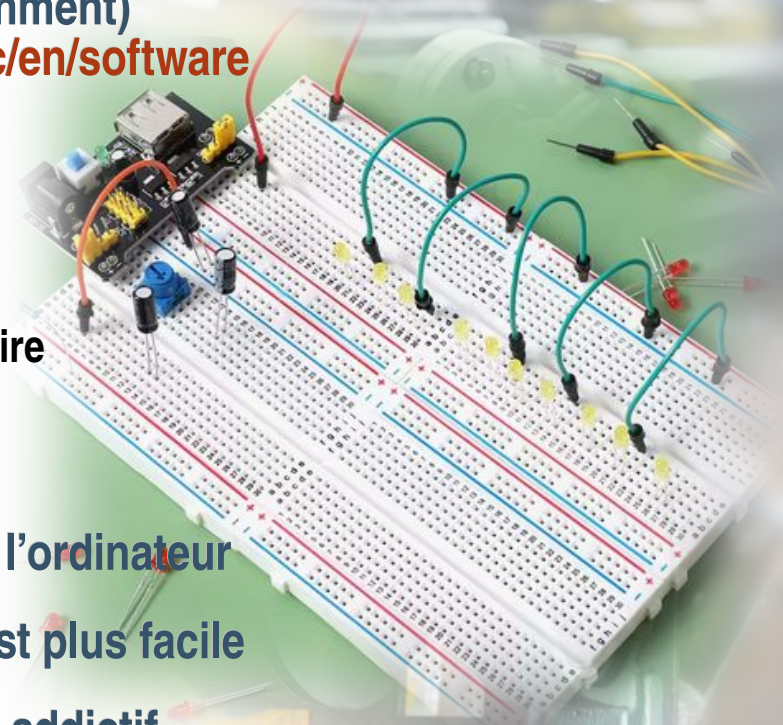
- L'IDE (Integrated Development Environment) Arduino …
  - Est un logiciel open source à installer sur un ordinateur
  - Est une interface qui permet
    - De rédiger les programmes
    - De générer le code machine et de le téléverser sur le module Arduino (via un câble USB)
  - Est basé sur le compilateur open source GCC (langage C/C++)
    - Compilation = « traduction » du texte écrit en respectant la syntaxe C/C++ en codes machine exécutables par le microcontrôleur cible
    - Il est nécessaire de préciser la carte Arduino avec laquelle on travaille avant compilation et téléchargement (génération du bon code machine compréhensible par le microcontrôleur cible)
  - Une option permet de choisir à quel microcontrôleur on s'adresse



IDROLIK
AUTOMATISATION HYDRAULIQUE

Démarrage avec Arduino

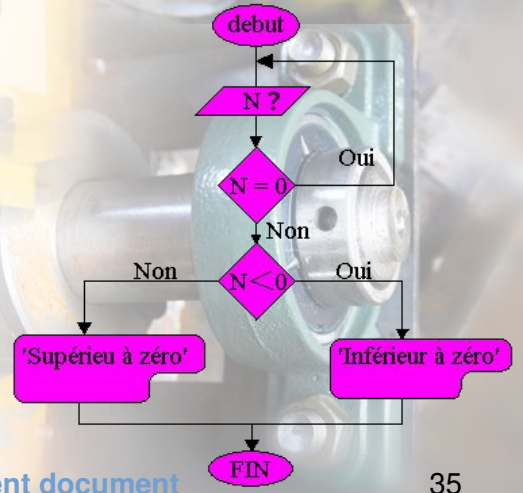
- Il faut un ordinateur (portable ou fixe)
- Il faut télécharger l'IDE (Integrated Development Environment) Arduino depuis le site Arduino : https://www.arduino.cc/en/software
  - Une fois l'IDE Arduino installé sur l'ordinateur, la liaison Internet n'est plus nécessaire
- Il faut acheter une carte Arduino, par exemple UNO R3
  - Si vous êtes sympa, vous l'achetez chez Arduino pour faire vivre la société et contribuer aux évolutions futures
  - Vous pouvez aussi acheter un clone chinois
- Il faut acheter un câble USB pour connecter l'Arduino à l'ordinateur
- Il faut acheter une plaque d'essai et des fils d'essai, c'est plus facile
- Il faut l'envie d'essayer. Attention, Arduino peut devenir addictif



• 3 - L’interface de programmation

Écrire un programme

- C'est être clair sur ce qu'on veut automatiser avec le microcontrôleur
  - Un organigramme et/ou un descriptif rédigé avec des phrases sont deux bon moyens de se fixer un cahier des charges (= contrat fonctionnel)
  - « Ce qui se conçoit bien s'énonce clairement et les mots pour le dire s'énoncent clairement »
    Ça va sans dire, mais ça va mieux en le disant, j'insiste
  - Question structurante liée au microcontrôleur : quelles informations va-t-on lire (entrées) et quelles commandes va-t-on générer (sorties) ?
- Une bonne pratique : écriture et test de programmes élémentaires correspondant chacun à un couple brique matérielle + logiciel
  - Choix et tests unitaires des capteurs que l'on pense utiliser
  - Choix et tests unitaires des actionneurs pressentis avec leur électronique de puissance (une sortie du microcontrôleur délivre quelques milliampères)
  - Ça veut dire qu'il faut identifier des matériels candidats et étudier leur datasheets. Datasheet pas claire = A FUIR.
    Trop de fonctions intégrées = A FUIR
    Capteurs compliqués à utiliser = A FUIR



Écrire un programme

- **Une bonne pratique : penser dès le début à la maintenabilité**
  - **On est très content de s'y retrouver des années après**
    - **Lorsqu'on veut améliorer et/ou étendre le programme**
    - **Lorsqu'un dispositif électronique qui n'est plus fabriqué tombe en panne et qu'on doit en utiliser un autre**
  - **Démarche et logique doivent rester claires et lisibles (= pas de bidouillage)**
    - **On commente abondamment : ce qui semble naturel lorsqu'on a la tête dans le projet ne l'est plus forcément des années après**
    - **On évite les pirouettes qui font gagner trois lignes de code : la lisibilité devrait rester la priorité**
    - **On regroupe ce qui va ensemble (données et fonctions) dans des objets (= classes) aussi indépendants que possible les uns des autres**
  - **L'interface avec chaque élément matériel devrait se limiter à un seul objet logiciel (= une classe)**
    - **Le remplacement d'un capteur ou d'un actionneur ne doit pas remettre en cause l'ensemble du programme (impact aussi localisé que possible)**
    - **Éviter les matériels multifonction : un matériel pour une fonction, c'est bien**

Écrire un programme

- Apprendre à se servir de l'IDE
  - Écrire du texte dans la fenêtre
  - Lancer la compilation et le téléversement vers le microcontrôleur (presse-bouton)
- Écrire un texte qui respecte la syntaxe du C++ :
  - Un programme est une suite d'instructions en C++ qui produit le fonctionnement attendu
- Trouver et corriger les erreurs de compilation (= instruction C++ → codes machine)
  - Le compilateur détecte des variables non initialisées, des incohérences de types, des dépassements de capacité ...etc… dont l'origine n'est pas toujours facile à identifier
- Tester le fonctionnement et revenir sur le programme autant que de besoin (essai – erreur)
  - Le plan de test doit être aussi exhaustif que possible
  - Logiciel instable = irritant

Spécificités C++ Arduino

- **Avec Arduino, on ne peut pas :**
  - Dupliquer un processus (fonction « fork() ») : duplication du processus courant en deux processus indépendants vis-à-vis du système d'exploitation, chacun avec son pointeur d'exécution, sa pile, son espace mémoire, son contexte.
    Par exemple, une variable qui a reçu une valeur avant le fork() continue à vivre sa vie dans chaque processus (= avec une valeur possiblement différente dans chaque processus)
  - Faire appel au multi threading = traitements d'un même processus qui tournent indépendamment, ce qui permet d'exploiter le caractère multi cœurs du microprocesseur tout en partageant le même espace mémoire (= les mêmes variables)
- **Pourquoi ?**
  Parce que ces deux fonctions font appel au système d'exploitation de l'ordinateur
  - Sur un ordinateur, quand on ne peut plus augmenter la fréquence d'horloge d'un processeur (limitation liée à l'électronique), on essaie de paralléliser les traitements sur plusieurs processeurs et/ou sur plusieurs cœurs
- **Certaines fonctions proches du hardware sont spécifiques à l'Arduino**
  - Exemples : « digitalWrite() » pour mettre 0 ou 1 sur une patte initialisée en sortie numérique, « analogRead() » pour lire la tension sur une patte initialisée comme une entrée analogique...

Spécificités C++ Arduino

- Deux fonctions sont exécutées par le microcontrôleur
- La fonction « setup() »
  - Exécutée une fois au démarrage du microcontrôleur
- La fonction « loop() »
  - Exécutée en boucle tant que le microcontrôleur est sous tension
- Une fois le programme implanté ...
  - Le microcontrôleur n'a plus besoin du câble USB (sauf si on souhaite s'en servir comme alimentation)
  - A chaque mise sous tension du microcontrôleur, « setup() » est exécuté, puis « loop() » est exécuté en boucle

```
class LED { // declaration d'une classe
  public :
    boolean etat_LED; // LED allumee ou eteinte
    LED(boolean etat_initial) { // Constructeur
      etat_LED = etat_initial;
    }
    void change () {
      if (etat_LED == true) {
        etat_LED = false;
      }
      else {
        etat_LED = true;
      }
    }
};

LED ma_LED = new LED(false); // instanciation
```

```
void setup() {
  /* setup() = fonction appelee une fois par au
     demarrage */
  pinMode(3, OUTPUT); // la patte 3 est une sortie
}
```

```
void loop() {
  /* loop() = fonction appelee en boucle par le
     microcontroleur */
  delay(1000); // attente 1000 ms = 1 s
  ma_LED.change();
  digitalWrite(3, ma_LED.etat_LED);
}
```

Affichage moniteur série

- Lorsque la carte Arduino est connectée à l'ordinateur via le câble USB …
  - Elle est alimentée électriquement via l'USB
  - Elle reçoit son code machine de l'ordinateur
  - Des instructions permettent de renvoyer des caractères vers la console de l'IDE Arduino via la liaison USB
- Les commandes relatives au moniteur série
  - Très utile pour rechercher des erreurs
  - Ouvrir la fenêtre du moniteur série de l'IDE dans « Outils »
  - « Serial.begin(9600) ; » : initie la communication série à 9600 bits / s (à placer dans « setup() »)
  - « Serial.println("Hello World !") ; » : écrit « Hello World ! » avec un saut de ligne à la fin

IDROLIK
AUTOMATISATION HYDRAULIQUE

C++

- **Commentaires :**
  - Dès qu'on trouve « // », le reste de la ligne est ignoré par le compilateur
  - Tout ce qui est encadré par « /* » et « */ » est ignoré par le compilateur
- **Un « ; » indique la fin d'une instruction**
  - Plusieurs instructions peuvent se suivre sur une même ligne, le séparateur est le « ; » mais ...
  - Si on veut rendre le programme illisible, on met plusieurs instructions à la suite sur la même ligne
- **Les blocs sont encadrés par « { } »**
- **Une bonne pratique : « outils » + « formatage automatique »**
  - En user et en abuser

```
class LED {  // declaration d'une classe
  public :
    boolean etat_LED;  // LED allumee ou eteinte
    LED(boolean etat_initial) {  // Constructeur
      etat_LED = etat_initial;
    }
    void change () {
      if (etat_LED == true) {
        etat_LED = false;
      }
      else {
        etat_LED = true;
      }
    }
};

LED ma_LED = new LED(false);  // instanciation

void setup() {
  /* setup() = fonction appelee une fois par au
     demarrage */
  pinMode(3, OUTPUT);  // la patte 3 est une sortie
}

void loop() {
  /* loop() = fonction appelee en boucle par le
     microcontroleur */
  delay(1000);  // attente 1000 ms = 1 s
  ma_LED.change();
  digitalWrite(3, ma_LED.etat_LED);
}
```

Variables en C++

- En C++, on déclare les variables qu'on va utiliser
  - Une variable permet de stocker un type d'information
  - Les variables peuvent être de plusieurs types
    - Booléen (vrai ou faux), Entier, Réel (simple ou double précision), Caractère
      int compteur; compteur = 12 ;
      float largeur ; largeur = 32,7 ;
      char carac ; carac = 'c' ; carac = 99 ;
    - Classes = objets composites
  - Un nom de variable est une suite de caractères (a, titi3, perimetre33, h_2_v …)
    - Pas de blancs (mettre des « _ » à la place si on veut un séparateur), pas de caractères accentués, pas de chiffre en première position, pas d'opérateurs… Globalement, pas de fantaisie, ça évite les ennuis
- Bonne pratique : déclarer toutes les variables en début de programme
  - Dans un fichier à part si besoin (directive de préprocesseur « #include nom_fichier »)

```
class LED { // declaration d'une classe
  public :
    boolean etat_LED; // LED allumee ou eteinte
    LED(boolean etat_initial) { // Constructeur
      etat_LED = etat_initial;
    }
    void change () {
      if (etat_LED == true) {
        etat_LED = false;
      }
      else {
        etat_LED = true;
      }
    }
};

LED ma_LED = new LED(false); // instanciation
```

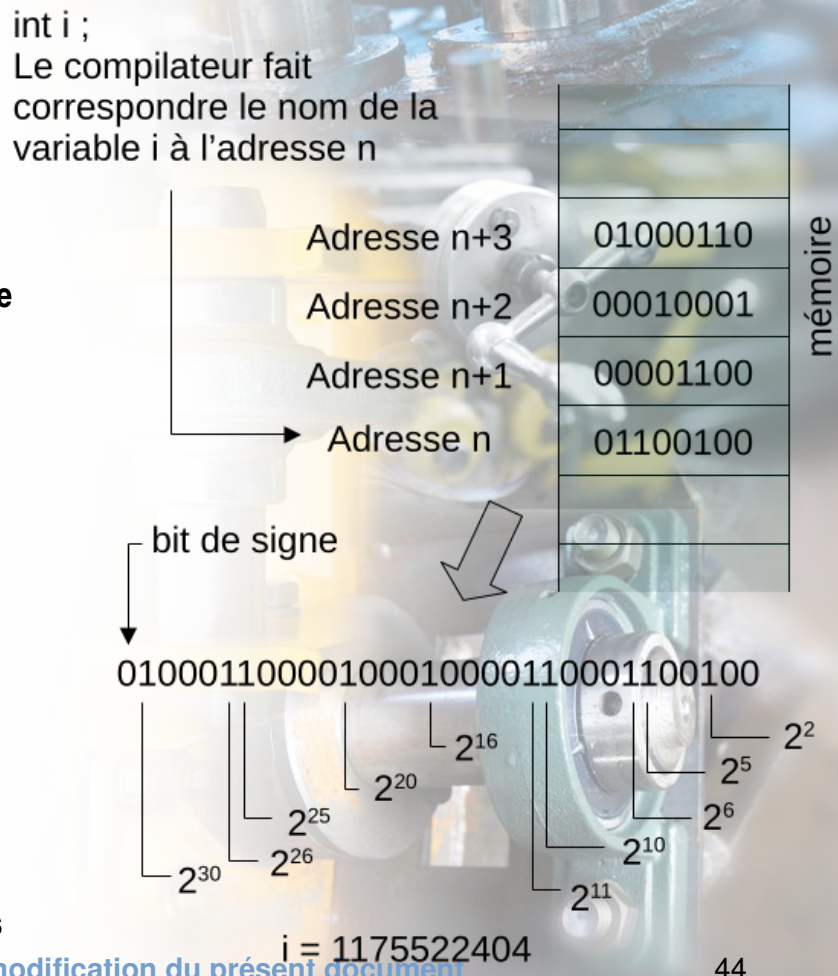
```
void setup() {
  /* setup() = fonction appelee une fois par au
     demarrage */
  pinMode(3, OUTPUT); // la patte 3 est une sortie
}

void loop() {
  /* loop() = fonction appelee en boucle par le
     microcontroleur */
  delay(1000); // attente 1000 ms = 1 s
  ma_LED.change();
  digitalWrite(3, ma_LED.etat_LED);
}
```

IDROLIK
AUTOMATISATION HYDRAULIQUE

- **Petit tour sur les variables**
  - **L'espace mémoire est compté, donc on compte**

Entier

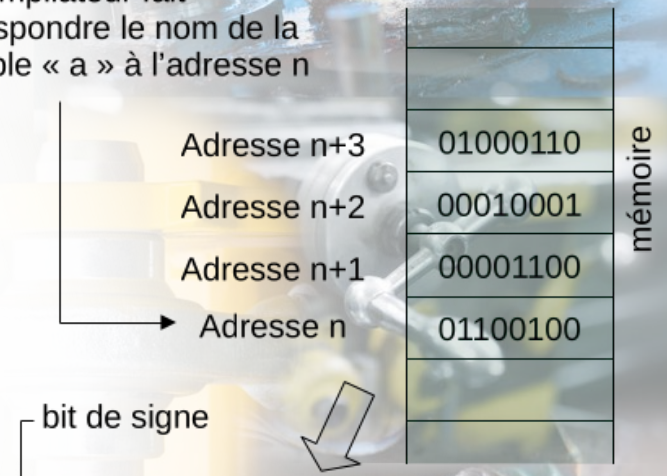
- **Déclaration d'une variable entière : « int i ; » ou « long i »**
  - La déclaration « long i » ; correspond à un entier long
  - Dans le code machine généré par le compilateur, une zone est réservée en mémoire (4 octets dans l'exemple ci-contre, 1 octet = 8 bits)
  - A chaque fois qu'on affecte une valeur à i, on modifie cette zone mémoire
- **Un entier codé sur 4 octets peut représenter un entier entre $-(2^{31} - 1)$ et $+(2^{31} - 1)$**
  - Dans la réalité, la valeur négative est codée en complément à deux (inversion bit à bit +1, dépassement ignoré).
    Ou comment transformer une soustraction en addition
- **En C++ on peut définir un pointeur vers un entier avec la déclaration « int *p ; »**
  - int i, *p ; // declaration d'un entier i et d'un pointeur p vers un entier
    p = &i ; // l'expression « &i » est l'adresse de i
    *p = 12 ; // l'adresse pointee par p reçoit 12
    // i vaut maintenant 12
  - Note : un tableau de pointeurs peut être très efficace pour trier des données volumineuses. L'intérêt des pointeurs est limité pour les entiers



Réel

- **Déclaration d'une variable réelle : « float a ; » ou « double a; »**
- **Un réel est codé sur 4 octets avec un signe, un exposant et une mantisse**
  - **Signe du réel : 1 bit (0 = positif)**
  - **Exposant : 8 bits, valeur positive à laquelle on retire systématiqement 127 → exposant positif ou négatif**
  - **Mantisse : 23 bits (bits derrière « 1, » implicite)**
- **Exemple : 5,75 en décimal s'écrit 101.11 en binaire**
  - **En base 10, chaque décimale divise la précédente par 10**
    0,1 = 1/10
    0,01 = 1/100
  - **En binaire, chaque « décimale$_{base\ 2}$ » divise la précédente par 2**
    0,1$_{base\ 2}$ = 1/2 = 0,5
    0,01$_{base\ 2}$ = 1/4 = 0,25

« float a ; »
Le compilateur fait correspondre le nom de la variable « a » à l'adresse n



01000110000100010000110001100100

Exposant
$= 2^7 + 2^3 + 2^2$
$= 140$
On enlève 127
Exposant = 13

Mantisse

$a = 1.00100010000110001100100 \times 2^{1101}$
$= 1001000100011.00011001_{base\ 2}$
$= 9285,09765625$

IDROLIK
AUTOMATISATION HYDRAULIQUE

Caractère

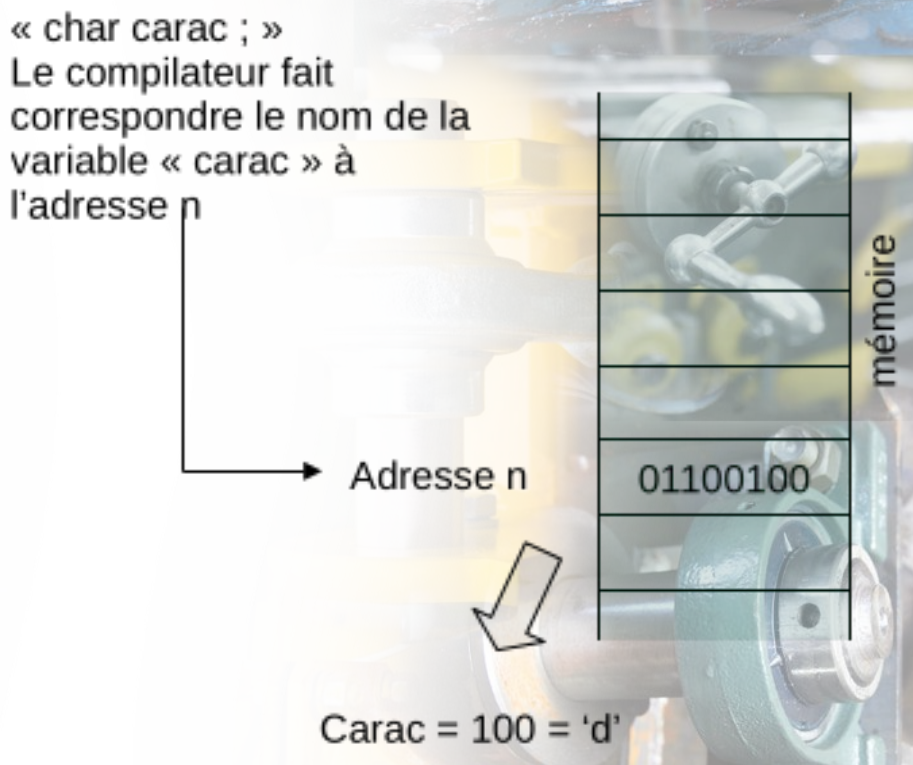
- Chaque caractère est associé à un code sur 7 bits
  - Le 8eme bit peut être utilisé comme bit de parité (= bit de contrôle) ou pour extension
- ASCII (American Standard Code for Information Interchange)
  - De 0 à 31 inclus : codes de contrôle
    ATTENTION : pas forcément visibles à l'affichage si présents
  - Exemple : les lignes des fichiers textes se terminent par un retour-chariot (13) et un line-feed (10).
    (références à nos ancêtres les machines à écrire)
- Bonne pratique :
  - Rester sur les caractères du code ASCII (32 ≤ c < 127)
  - Éviter les caractères accentués (codes > 127) et les jeux de caractères spécifiques à chaque pays

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Caractère

- Un caractère consomme un octet en mémoire
  - On verra plus tard qu'on peut stocker une chaîne de caractères sous la forme d'un tableau de caractères
- Note : un booleen (vrai ou faux) est stocké comme un caractère
  - 0 = faux
  - Différent de 0 = vrai



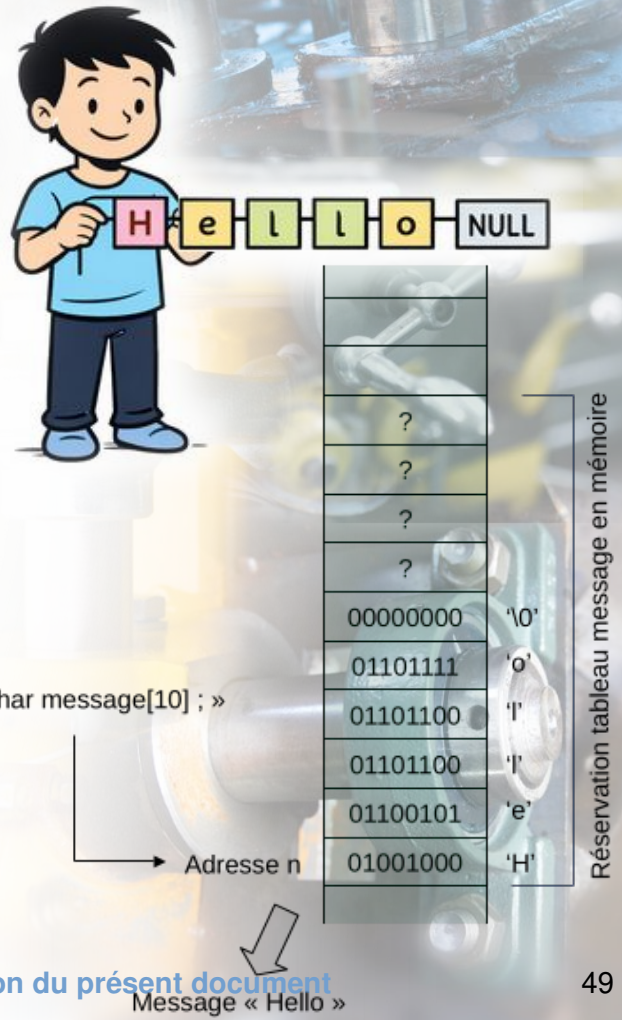
Tableaux

- Chaque type de variable peut donner lieu à un tableau
  - Utilisé pour manipuler un ensemble de données de même nature (éléments numérotés)
  - Pour chaque tableau, on indique le nombre d'éléments
  - Un tableau commence à 0
  - Chaque élément d'un tableau est utilisé comme une variable
- Tableau d'éléments de type « class »
  - Plus simple si le constructeur de la classe n'attend pas d'arguments
  - Commode pour manipuler les informations d'une collection d'objets similaires (capteurs, moteurs pas-à-pas ...)

```
int i, Toto[10];

for (i = 0; i < 9; i++) {
      Toto[i] = 0;
}


class LED { // declaration d'une classe
  public :
    boolean etat_LED; // LED allumee ou eteinte
    LED() { // Constructeur
      etat_LED = false;
    }
    void change ()
    {
      if (etat_LED == true) { // == comparaison
        etat_LED = false; // = affectation
      }
      else {
        etat_LED = true;
      }
    }
};

LED ma_LED[5];

for (i = 0; i < 5; i++) {
    ma_LED[i].change();
  }
```

Chaîne de caractères

- Souvent, on veut afficher des messages, sous la forme de chaînes de caractères (par exemple l'heure, ou une température, ou encore un message d'erreur)
- Il existe une classe « String » qui permet de manipuler des chaînes de caractères
  - Cette classe s'occupe de réserver dynamiquement de l'espace mémoire
- Perso, je préfère la méthode classique qui traite les chaînes de caractères comme des tableaux de caractères
  Pourquoi ? Parce que j'aime bien maîtriser la quantité de mémoire utilisée
  - La fin de la chaîne de caractères est identifiée par un caractère NULL (= 0) qui suit le dernier caractère de la chaîne
  - Bien sûr, le caractère NULL doit faire partie de la réservation en mémoire
  - Exemple : la chaîne de caractères « Hello » nécessite la réservation en mémoire d'un tableau de caractères d'au moins 6 éléments
  - char message[10] ; // reservation pour une chaine de 9 caracteres max
    message[0] = 'H' ; // elements du tableau numerotes a partir de 0
    message[1] = 'e' ;
    ...
    message[5] = 'o' ;
    message[6] = '\0' ; // caractere NULL pour marquer la fin de la chaine



Chaîne de caractères

- La commande magique : « sprintf() ; »

```
char b[11], c[51] ;
int age ;
float poids, taille ;

strcpy(b, "je pese") ;
age = 60 ;
poids = 86.3 ;
taille = 1.8 ;

sprintf(c, "J'ai %d ans,%s %f kg, taille : %f m",
                                    age, b, poids, taille) ;
```

- Exemple ci-dessus :
  - Écrit dans le tableau « c » la chaîne de caractères "J'ai 60 ans, je pese 86,3 kg, taille : 1,8 m"
  - L'entier 60 est devenu deux caractères : '6' et '0'
  - Le caractère '\0' est ajouté en fin de tableau

Classe

- Une classe permet de regrouper des variables et des méthodes (fonctions)
  - Les variables peuvent être des entiers, des réels, des caractères, des tableaux, des instanciations d'autres classes …
  - Les méthodes sont des fonctions
    - Une fonction exécute un ensemble d'instructions
    - Une fonction peut attendre des arguments (de différents types) … ou pas.
    - Une fonction peut retourner une valeur … ou pas (void).
    - Une fonction se déclare sous la forme : « int ma_fonction(int i, float a, …) { instructions… } »
- C'est un excellent moyen d'organiser et de manipuler des objets

Classe

- Pour une classe (ici « LED »), une zone mémoire est réservée pour les éléments constants.
  Elle est partagée par toutes les instances
  - En général, les éléments constants sont du code, on peut aussi définir des constantes (comme π)
  - Ici, un espace mémoire pour le code de la fonction « LED () »
    - Fonction de même nom que la classe = constructeur (exécuté à la création de chaque instanciation)
  - Ici, un autre espace mémoire pour le code de la fonction « change() »
- Pour chaque instance d'une classe (ici « ma_LED »), un espace mémoire indépendant est réservé pour les variables de chaque instanciation de « LED »
  - Ici : « etat_LED »

```
class LED { // declaration d'une classe
  public :
    boolean etat_LED; // LED allumee ou eteinte
    LED(boolean etat_initial) { // Constructeur
      etat_LED = etat_initial;
    }
    void change () {
      if (etat_LED == true) {
        etat_LED = false;
      }
      else {
        etat_LED = true;
      }
    }
};

LED ma_LED(false); // instanciation

void setup() {
  /* setup() = fonction appelee une fois par au
     demarrage */
  pinMode(13, OUTPUT); // la patte 13 est une sortie
}

void loop() {
  /* loop() = fonction appelee en boucle par le
     microcontroleur */
  delay(1000); // attente 1000 ms = 1 s
  ma_LED.change();
  digitalWrite(13, ma_LED.etat_LED);
}
```

Classe - héritage

- Je peux créer une classe à partir d'une autre
  - La nouvelle classe hérite des attributs et des méthodes de la classe mère
  - Je peux l'enrichir avec des nouveaux attributs et des nouvelles méthodes (surcharge)
  - Je peux réécrire des méthodes en gardant leur nom à condition qu'elles aient les mêmes arguments et la même valeur de retour (redéfinition)
- Note :
  - En C++, la méthode « int addition(int, int) » (qui ajoute deux entiers et retourne un entier) et une méthode « vecteur addition(vecteur, vecteur) » (qui ajoute deux vecteurs et retourne un vecteur) sont deux méthodes distinctes pour le compilateur
  - On peut donc faire cohabiter plusieurs fonctions qui portent le même nom si les arguments sont différents

```
class Vehicule {
}

class Voiture : public Vehicule {
// Une Voiture EST UN Vehicule
}

class Moto : public Vehicule {
// Une Moto EST AUSSI un Vehicule
}
```

Pointeurs

- En C++, pour chaque type, on peut définir des pointeurs
  - Exemple :
    int i ; // i est une variable entière
    int *p ; // p est un pointeur vers une variable entière
    p = &i ; // p reçoit l'adresse de i
    *p = 4 ; // écrit 4 dans i
  - Le pointeur conserve en mémoire … une adresse mémoire
- Utile lorsqu'on veut trier par remontée de bulle des objets qui occupent beaucoup d'espace mémoire
  - Tri par remontée de bulle : comparaisons deux à deux des objets et inversion si mal placés
  - Espace mémoire occupé par une instance de classe « mon_objet » = « sizeof(mon_objet) »
  - Il est plus aisé d'intervertir les contenus de deux pointeurs en mémoire que d'intervertir le contenu de deux objets occupant chacun beaucoup d'espace mémoire
- On peut aussi définir des pointeurs vers des fonctions
  - Une fonction est du code machine stocké en mémoire
  - Un pointeur vers ce code contient l'adresse du code de la fonction
  - Utile lorsqu'on veut demander à une fonction d'utiliser la (les) fonction(s) fournie(s) en argument(s)

IDROLIK
AUTOMATISATION HYDRAULIQUE

- **Petit tour sur les opérations de base**
  - **Sans ça, pas de programme**

Les opérations de base

- De base, le microcontrôleur sait
  - Aditionner (« + »), soustraire (« - »), multiplier (« * »), diviser (« / »)
- Ces opérations peuvent s'appliquer
  - Aux entiers (« int » ou « long »), aux réels (« float » ou « double »), aux caractères (« char »)
- Si une opération dépasse la capacité des variables
  - Vous aurez un résultat arrondi (exemple l'entier 11 divisé par 2)
  - Ou une erreur de compilateur ou d'exécution (exemple : division par zéro)
  - Ou encore un déroulement sans erreur d'exécution mais avec un résultat potentiellement non attendu (exemple soustraction de $1.25 \times 10^{15} - 1{,}37 \times 10^{-15}$)
- Pour d'autres opérations, il faudra utiliser des fonctions
  - Soit des fonctions toutes faites (exemple « pow(x,y) » pour x^y)
  - Soit des fonctions que vous aurez faites vous-mêmes

```
// factorielle(5) = 1 x 2 x 3 x 4 x 5
int factorielle(int n) {
  int k ;
  if (n < 1) {
    return(-1) ; // erreur
  }

  if (n == 1) {
    return(n) ;
  }
  else { // n est superieur a 1
    k = n * factorielle(n - 1) ;
    return(k) ;
  }
}
```

Les tests

- « if (condition) { instructions … } »
- « if (condition) { instructions… }
  else { instructions … } »
- Condition :
  - « (a > b) », « (a < b) », « (a == b) »
    « (a <= b) », « (a >= b) », « (a != b) »
    « (strcmp(s1,s2) == 0) »
  - Fonction qui retourne un booléen (vrai ou faux)
  - Attention : « = » affectation, « == » comparaison
- Condition1 ou condition 2 : « || »
  - if ((a > b) || (c < 3)) { instructions … }
- Condition 1 et condition 2 : « && »
  - if ((a > b) && (c < 3)) { instructions … }
- On peut mixer :
  - If ((a > b) || ((c > d) && (e == 0))) { instructions … }

```
int a = 5;
int b = 8;
if (a < b):
    string meilleur_telephone = "bleu";
} else {
    string meilleur_telephone = "jaune";
}
```


```
void setup() {
   pinMode(5, INPUT); // la broche 5 est une entree
   pinMode(6, OUTPUT); // la broche 6 est une sortie
   }

void loop() {
   If (digitalRead(5) == HIGH) { // si 5V sur la broche 5
     digitalWrite(6, LOW); // on met la broche 6 a 0V
   } else {
     digitalWrite(6, HIGH); // sinon, on met la broche 6 a 5V
   }
}
```

Les boucles

- **Boucle « for »**
  - « for (i=0 ; i<10 ; i=i+1) { instructions ... } »
  - i commence à 0, on exécute les instructions tant que i < 10, i = i+1 avant de passer au tour suivant
- **Boucle « while »**
  - Tant que la condition est vraie on exécute le bloc d'instructions
  - « while (a > b) { instructions … }
- **Boucle « do while »**
  - Exécuter le bloc d'instructions et vérifier la condition avant de l'exécuter à nouveau
  - « do { instructions … } while (i == 0) ; »


```
int i;
boolean k[3];

void setup() {
  for (i = 3; i < 6; i++) {
    pinMode(i, INPUT); // la broche i est une entree
  } // les broche 3, 4 et 5 sont des entrees
  pinMode(6, OUTPUT); // la broche 6 est une sortie
}

void loop() {
  do {
    for (i = 3; i < 6; i++) {
      k[i - 3] = digitalRead(i); // lecture etat broche i
    }
  } while ((k[0] == LOW) && (k[1] == LOW) && (k[2] == LOW));
  digitalWrite(6, HIGH); // on met la broche 6 a 5V
  delay(1000); // on attend 1 s
  digitalWrite(6, LOW); // et on remet la broche 6 a 0V
}
```

Opérations bit à bit

- « & » : et
  - Permet par exemple de tester facilement si un bit d'un octet est à 1
  - Ou de mettre un bit d'un octet à 0
- « | » : ou inclusif
  - Permet par exemple de mettre un bit d'un octet à 1
- « ^ » : ou exclusif
- « ~ » : non
- « >> » : décalage à droite
  - char c ;
    c = c & 0b11111000 ; // met les
    // 3 bits de poids faible à 0
    c = c >> 3 ; // decale les 5 bits de gauche vers la droite
- « << » : décalage à gauche

Fonctions

- **Une fonction est une séquence de code destinée à être appelée régulièrement**
  - Le processeur exécute le code en cours (qui peut déjà être celui d'une fonction)
  - Il tombe sur un appel à une fonction
    - Avant d'aller exécuter le code en question, il pose l'adresse actuelle du pointeur d'exécution sur la pile
  - Positionnement du pointeur d'exécution au début du code de la fonction puis exécution
    - À la fin de l'exécution de la fonction, récupération de l'adresse de retour sur la pile
  - Poursuite de l'exécution du code initial
- **Une fonction peut attendre des arguments et retourner une valeur**
  - Déclaration : « float surface(float a, float b) { … } »
    Appel : « s = surface(12.5 , 45.8) ; »
  - Déclaration : « void allume_LED() { … } »
    Appel : « allume_LED() ; }


```
float surface(float largeur, float longueur) {
  float S ; // variable locale à la fonction

  S = largeur * longueur ;
  return(S) ;
}
```

Pointeurs de fonction

- Un pointeur peut être passé en argument à une fonction
  - La fonction peut alors écrire à l'adresse du pointeur …
  - … ou exécuter le code qui se trouve à l'adresse passée
- Déclaration d'un pointeur vers une fonction
  - « float (*toto)(float a, float b) ; »
    - Définit un pointeur toto qui peut recevoir l'adresse d'une fonction qui attend 2 arguments réels et retourne un réel
  - ATTENTION : une fonction « ordinaire » et une méthode d'une classe sont traitées différemment, et là, ça se complique
- La fonction peut lancer le traitement que lui a passé le code appelant
- Mon conseil :
  - Utiliser les pointeurs de fonctions avec parcimonie (complexifie l'écriture, surtout s'agissant des méthodes de classes)
  - Se débrouiller avec les valeurs retournées : plus simple et plus lisible.

Ou comment appeler une fonction en lui passant une action à exécuter

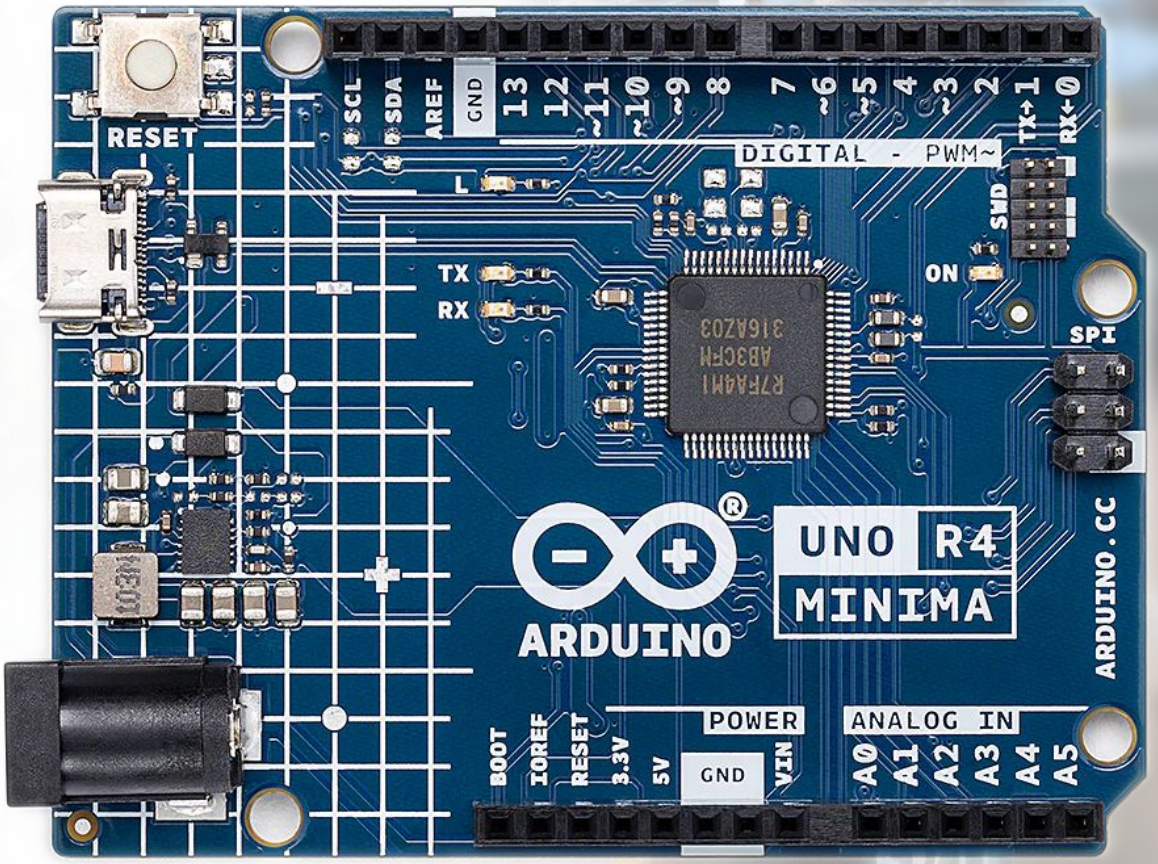
```
void teste_bouton(int pin, void (*action)()) {
  if (digitalRead(pin) == LOW) {
    action();
  }
}
```

- **Le bouton poussoir, la LED, l'afficheur 7 segments**
  - **Premiers programmes**
  - **Premiers branchements**

Bouton poussoir, LED

- On veut réaliser un montage avec :
  - Un bouton poussoir
  - Une LED
- Objectif du montage :
  - À chaque fois qu'on appuie sur le bouton poussoir ...
  - ... on fait clignoter 3 fois une LED
- Étape 1 :
  - On jette un coup d'œil sur la carte Arduino pour décider sur quelles broches on va brancher le bouton poussoir et la LED
  - On a besoin de deux broches numériques, on en a 14 à disposition (0 à 13)

    ►on choisit la broche 12 pour le bouton poussoir et la broche 13 pour la LED (deux broches qui ne savent faire que du numérique de base sur cette carte)

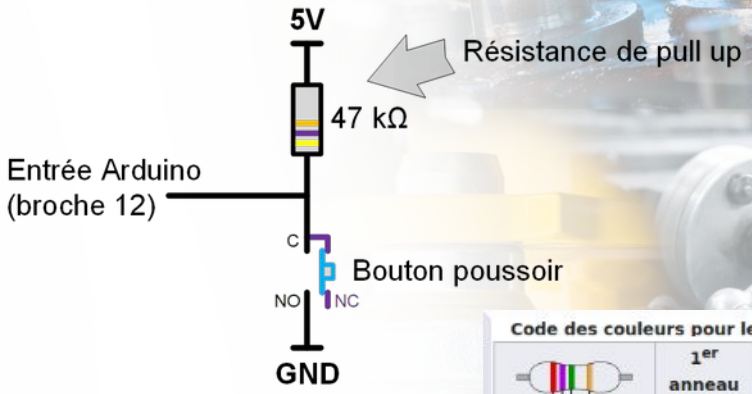


Résistance, U = R x I

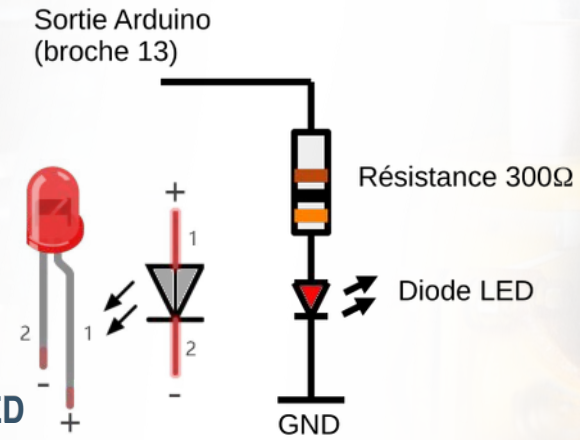
- **Analogie :**
  - Tension électrique = motivation à passer pour les électrons
    - Plus la tension est forte, plus les électrons veulent passer
  - Résistance électrique = gêne au passage des électrons
    - Plus la résistance est grande, plus elle s'oppose au passage des électrons malgré leur motivation à passer
  - Intensité = débit d'électrons qui passent
    - À tension et résistance données, une intensité (débit d'électrons) passe, c'est une conséquence
- **Tension électrique U, résistance R et intensité I sont liées par la loi « U = R x I »**
  - U en volts (V), R en ohms (Ω), I en ampères (A)
  - En général, on applique une tension (par exemple 5V). Si on place une résistance sur le passage du courant (par exemple 300Ω), ALORS il passe une intensité I = U / R (ici, 17 mA)

Connexion physique

- **Connexion du bouton poussoir**
  - Une résistance de pull-up force le 5V lorsque le bouton poussoir ne force pas le potentiel à zéro
    - Note : « pinMode(pin, INPUT_PULLUP) » active une résistance pull-up interne
  - Bouton poussoir appuyé ► entrée 12 = 0



- **Connexion de la LED**
  - On considère que notre LED passante va installer une tension d'1,2V à ses bornes
  - On veut faire passer 10 à 15 mA dedans
  - On va mettre une résistance en série avec la LED
  - U = R x i
    U tension en volts (3,8V pour nous)
    R : résistance en ohm (ce qu'on cherche)
    i : intensité ampère (0,0125 A pour nous)
  - R = U / i = 3,8 / 0,0125 = 300 Ω



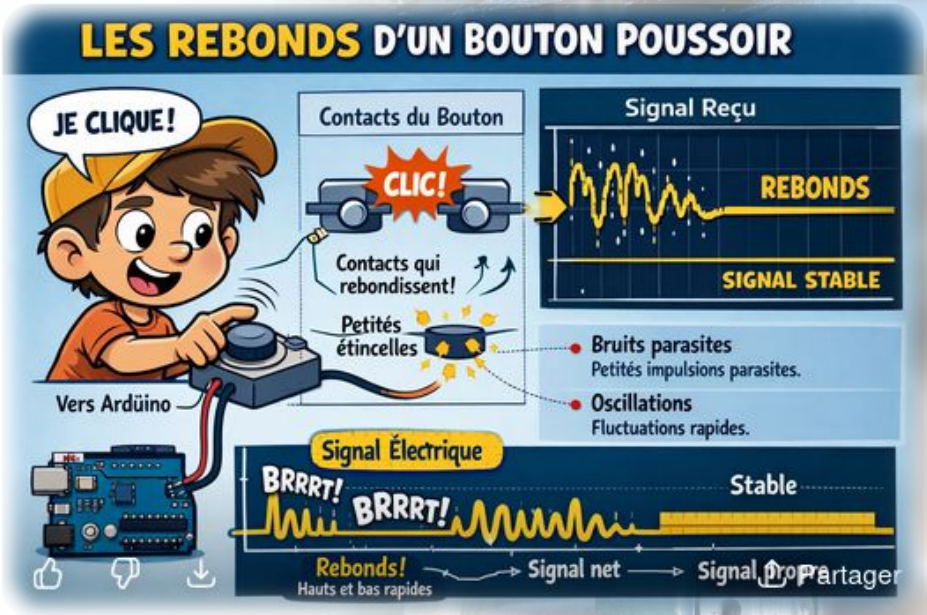
- **Note : sur bon nombre de cartes Arduino, une LED indique l'état de la pin 13**

Code des couleurs pour les résistances

	Couleur	1er anneau gauche	2e anneau gauche	Dernier anneau gauche	Anneau droite
		1er chiffre	2e chiffre	Multiplicateur	Tolérance
	noir	0	0	$10^0=1$	± 20 %
	marron	1	1	10^1	± 1 %
	rouge	2	2	10^2	± 2 %
	orange	3	3	10^3	
	jaune	4	4	10^4	
	vert	5	5	10^5	± 0,5 %
	bleu	6	6	10^6	± 0,25 %
	violet	7	7	10^7	± 0,10 %
	gris	8	8	10^8	± 0,05 %
	blanc	9	9	10^9	
	or			0,1	± 5 %
	argent			0,01	± 10 %
	(absent)				± 20 %

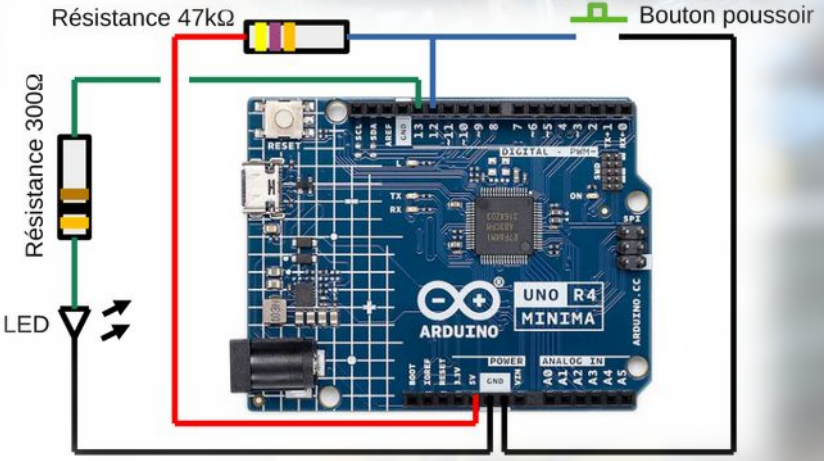
Rebonds bouton poussoir

- **Lorsqu'on utilise un contacteur mécanique ...**
  - On a des chances d'avoir des rebonds au niveau du contact
  - Pendant quelques millisecondes, le signal peut être instable
- **Le microcontrôleur fonctionne à plusieurs MHz**
  - Le microcontrôleur boucle suffisamment rapidement pour « voir » des états hauts et bas pendant les rebonds
  - La programmation doit en tenir compte
  - Lorsque l'état du bouton poussoir change (appui puis relâchement), ne pas tenir compte de ce qui se passe dans les millisecondes qui suivent
- **Bonne pratique :**
  - C'est un humain qui agit sur le bouton poussoir. Il est donc raisonnable de penser que même pour un excité du bouton poussoir, deux appuis successifs seront espacés de plus d'un quart de seconde
  - Il faut choisir à quel changement d'état on réagit (appui ou relâchement)
  - « Appui + relâchement rapide » et « appui + maintien + relâchement » peuvent être discriminés par le programme



Programme minimaliste

- Ce programme est simple et il fonctionne
  - Si on laisse le bouton appuyé, il recommence, c'est bien
- Pour autant il a un gros défaut : il est peu extensible
  - 1)- il utilise l'instruction « delay() » qui monopolise le microcontrôleur pour de l'attente
    - Qu'est-ce que je fais si j'ai besoin de surveiller un autre capteur pendant ce temps là ?
      Ou un autre bouton, ou encore générer une action ?
  - 2)- il ne traite pas le bouton poussoir comme une brique technologique
    - Test et validation de la brique technologique une bonne fois pour toutes
    - Réutilisation et maintenance simplifiées si tous les boutons poussoirs sont traités de la même manière




```
int i;

void setup() {
  pinMode(12, INPUT); // broche 12 : entree
  pinMode(13, OUTPUT); // broche 13 : sortie
}

void loop() {
  if (digitalRead(12) == LOW) { // si broche 12 a 0V
    for (i = 0; i < 3; i++) { // on clignote 3 fois
      digitalWrite(13, HIGH); // broche 13 a 5V
      delay(1000); // attente 1 seconde
      digitalWrite(13, LOW); // broche 13 a 0V
      delay(1000); // attente 1 seconde
    }
  }
}
```

Programme1

Pas extensible

Clignotement « passant »

- La fonction millis()
  - N'attend pas d'arguments
  - Retourne le nombre de millisecondes depuis la mise sous tension (unsigned long)
  - sizeof(unsigned long) = 4 octets
    $2^{32} - 1 = 4294967295$ ms = 49,7 jours
- Bonne pratique : rendre le programme « passant »
  - Si c'est le moment on agit, sinon on passe, mais on fait tout pour ne pas bloquer le déroulement
  - Par exemple :
    Faire clignoter une LED peut se résumer à un test à chaque boucle
    - Est-ce le moment de changer l'état de la LED ?
      Si oui, on change son état
      Sinon on verra au prochain tour
  - En faisant ça, chaque boucle permet de tester plusieurs capteurs et plusieurs boutons, tout en faisant clignoter la LED

```
unsigned long dernier_changt_LED, t;
boolean etat_LED;

void setup() {
  dernier_changt_LED = 0;
  pinMode(13, OUTPUT);
  etat_LED = LOW;
  digitalWrite(13, etat_LED);
}

void loop() {

  // ...

  // on s'occupe de la LED en passant
  // sans jamais bloquer la boucle
  t = millis();
  if ((t - dernier_changt_LED) >= 1000) {
    // 1000 ms se sont ecoulees
    // il est temps de changer l'etat de la LED
    if (etat_LED == HIGH) {
      etat_LED = LOW;
    } else {
      etat_LED = HIGH;
    }
    digitalWrite(13, etat_LED);
    dernier_changt_LED = t;
  }

  // ...
}
```

IDROLIK
AUTOMATISATION HYDRAULIQUE

- On revient sur notre objectif
  - Quand on appuie sur le bouton poussoir, la LED clignote 3 fois
  - On laisse le programme « passant »
- 2 tests traversés à chaque boucle à la vitesse du microcontrôleur
- Plus compliqué que le programme1 mais…
  - On peut surveiller autre chose
  - Par exemple on peut envisager plusieurs boutons poussoirs
  - Chaque bouton peut faire clignoter sa LED à un rythme qui lui est propre

Programme2

```
unsigned long dernier_changt_LED, t;
boolean etat_LED, clignote;
int compteur;

void setup() {
  dernier_changt_LED = 0; t = 0; compteur = 0;
  pinMode(12, INPUT_PULLUP);
  pinMode(13, OUTPUT);
  etat_LED = LOW;
  digitalWrite(13, etat_LED);
  clignote = false;
}

void loop() {
  if ((clignote == false) && (digitalRead(12) == LOW)) {
    clignote = true;
    compteur = 0;
  }

  if (clignote == true) { // on s'occupe de la LED
    t = millis();
    if ((t - dernier_changt_LED) >= 1000) { // changt LED
      if (etat_LED == HIGH) {
        etat_LED = LOW;
        if (compteur >= 3) { // on vient d'eteindre la LED
          clignote = false; // et on a clignote 3 fois
        }
      } else {
        etat_LED = HIGH;
        compteur ++;
      }
      digitalWrite(13, etat_LED);
      dernier_changt_LED = t;
    }
  }
}
```

Mieux

IDROLIK
AUTOMATISATION HYDRAULIQUE

Même chose en créant une classe LED

Programme3

```
class LED {
  public :
    unsigned long dernier_changt_LED;
    boolean etat_LED, clignote;
    int compteur, pin, demie_periode;
```

Variables

```
    LED () { // lancement auto
             // pour chaque instance
      dernier_changt_LED = 0;
      compteur = 0;
      clignote = false;
      demie_periode = 0;
    }
```

Constructeur

```
    void init_OUTPUT(int i, int j) {
      pin = i;
      demie_periode = j;
      pinMode(pin, OUTPUT);
      etat_LED = LOW;
      digitalWrite(pin, etat_LED);
    }
```

Fonction 1

(suite classe LED)

```
    void demarre_clignotement() {
      if (clignote == false) {
        clignote = true;
        compteur = 0;
      }
    }
```

Fonction 2

```
    void gere_en_passant() {
      unsigned long t;
```

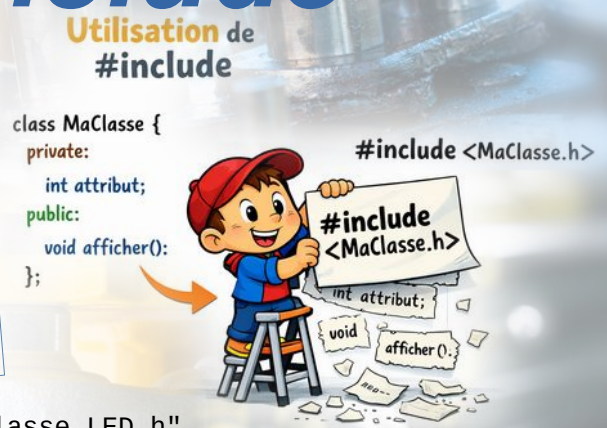
t : variable locale

```
      if (clignote == true) { // on s'occupe de la LED
        t = millis();
        if ((t - dernier_changt_LED) >= demie_periode) {
          if (etat_LED == HIGH) {
            etat_LED = LOW;
            if (compteur >= 3) { // on vient d'eteindre la LED
              clignote = false; // et on a clignote 3 fois
            }
          } else {
            etat_LED = HIGH;
            compteur ++;
          }
          digitalWrite(pin, etat_LED);
          dernier_changt_LED = t;
        }
      }
    }
};
```

Fonction 3

Directive « #include »

- La classe étant définie
  - Le corps du programme est simplifié
  - Extension facilitée
    - Exemple : plusieurs boutons poussoirs qui font chacun clignoter 3 fois une LED à des fréquences différentes
- Directive « #include "fichier" »
  - Le préprocesseur intègre le contenu du fichier comme s'il avait été écrit à cet endroit
  - Allège visuellement le programme
- Note :
  - « #include "mon_fichier" » cherche mon_fichier à coté du programme (en général « mon_fichier.h », « h » pour header
  - « #include <fichier> » cherche le fichier dans les répertoires où se trouvent les bibliothèques standards



Programme3 suite et fin

```
#include "ma_classe_LED.h"

LED ma_premiere_led; // creation instance avec
                     // lancement constructeur


void setup() {
  pinMode(12, INPUT_PULLUP);
  ma_premiere_led.init_OUTPUT(13, 1000);
}


void loop() {
  if (digitalRead(12) == LOW) {
    ma_premiere_led.demarre_clignotement();
  }

  ma_premiere_led.gere_en_passant();
}
```

Encore mieux

IDROLIK
AUTOMATISATION HYDRAULIQUE

Programme4 « fichier_LED.h »

- **Version finale « bouton poussoir & clignotement »**
  - **Programme « passant »**
  - **Utilisation de classes**
    - **Classe LED**
    - **Classe BOUTON_POUSSOIR**

```
class LED {
  public :
    unsigned long dernier_changt_LED;
    boolean etat_LED, clignote;
    int compteur, pin, nbre_clignotements, demie_periode;

    LED () { // constructeur
      dernier_changt_LED = 0; compteur = 0;
      clignote = false;
      nbre_clignotements = 0; demie_periode = 0;
    }

    void init_OUTPUT(int i) {
      pin = i;
      pinMode(pin, OUTPUT);
      etat_LED = LOW;
      digitalWrite(pin, etat_LED);
    }

    void demarre_clignotement(int i, int j) {
      if (clignote == false) {
        clignote = true;
        nbre_clignotements = i;
        demie_periode = j;
        compteur = 0;
        etat_LED = LOW;
      }
    }

    void gere_en_passant() {
      unsigned long t;

      if (clignote == true) { // on s'occupe de la LED
        t = millis();
        if ((t - dernier_changt_LED) >= demie_periode) {
          if (etat_LED == HIGH) {
            etat_LED = LOW;
            if (compteur >= nbre_clignotements) {
              clignote = false;
            }
          } else {
            etat_LED = HIGH;
            compteur ++;
          }
          digitalWrite(pin, etat_LED);
          dernier_changt_LED = t;
        }
      }
    }
};
```

IDROLIK
AUTOMATISATION HYDRAULIQUE

Programme4 « fichier_BOUTON_POUSSOIR.h »

- **Classe « BOUTON_POUSSOIR »**
  - **Gère appui court et appui long**
  - **Démarre le clignotement au relâchement du bouton**

```
class BOUTON_POUSSOIR {
  public:
    unsigned long date_dernier_changement, duree_rebonds;
    unsigned long duree_mini_appui_long;
    int pin;
    boolean appui;

    BOUTON_POUSSOIR() {
      date_dernier_changement = 0; duree_rebonds = 250;
      duree_mini_appui_long = 1000; pin = 0; appui = false;
    }

    void init_INPUT(int i, unsigned long j, unsigned long k) {
      pin = i;
      duree_rebonds = j;
      duree_mini_appui_long = k;
      pinMode(pin, INPUT_PULLUP);
    }

    char gere_en_passant() {
      unsigned long t;
      char carac_retour;

      carac_retour = ' ';
      t = millis();
      if ((t - date_dernier_changement) >= duree_rebonds) {
        // Si on n'est plus dans les rebonds ...

        if ((appui == false) && (digitalRead(pin) == LOW)) {
          // bouton presse alors que le bouton etait relache
          appui = true;
          date_dernier_changement = t;
        } else {
          if ((appui == true) &&
                                  (digitalRead(pin) == HIGH)) {
            // bouton relache alors qu'il etait presse
            appui = false;
            if ((t - date_dernier_changement) <
                                        duree_mini_appui_long) {
              carac_retour = 'C'; // appui court
            } else {
              carac_retour = 'L'; // appui long
            }
            date_dernier_changement = t;
          }
        }
      }
      return (carac_retour);
    }
};
```

IDROLIK
AUTOMATISATION HYDRAULIQUE

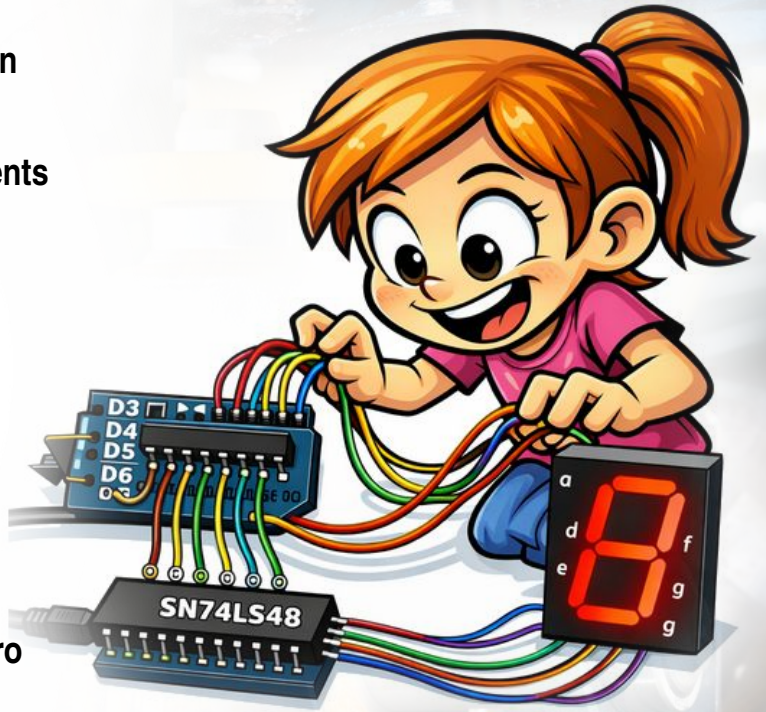
Corps du programme4 (suite et fin)

- **Version finale**
  - **Appui court : clignote 3 fois lentement**
  - **Appui long : clignote 10 fois rapidement**
  - **Gère la problématique des rebonds**
  - **Rend aisée l'utilisation de plusieurs boutons poussoirs qui font clignoter simultanément plusieurs LEDs à des rythmes différents**
  - **Programme « passant » qui n'empêche pas la surveillance d'autres capteurs**


```
#include "fichier_LED.h"
#include "fichier_BOUTON_POUSSOIR.h"

LED ma_premiere_led; // creation instance
BOUTON_POUSSOIR mon_premier_bp; // creation instance

void setup() {
  pinMode(12, INPUT_PULLUP);
  ma_premiere_led.init_OUTPUT(13);
  mon_premier_bp.init_INPUT(12, 250, 1500);
}

void loop() {
  char c;

  c = mon_premier_bp.gere_en_passant();
  if (c == 'C') {
    ma_premiere_led.demarre_clignotement(3, 1000);
  } else {
    if (c == 'L') {
      ma_premiere_led.demarre_clignotement(10, 500);
    }
  }

  ma_premiere_led.gere_en_passant();
}
```

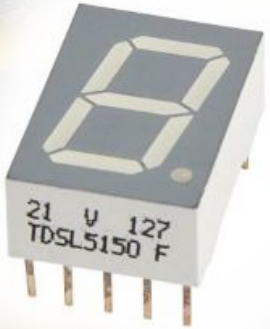
L'afficheur 7 segments

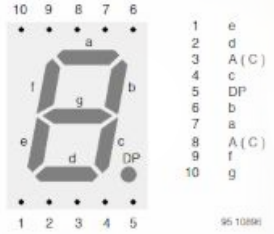
- **Afficheur 7 segments**
  - On peut utiliser 4 sorties binaires de l'Arduino pour commander un afficheur 7 segments
  - ► l'Arduino commande 4 entrées d'un décodeur BCD-to-7-segments
  - ► le décodeur commande les 7 segments de l'afficheur à LEDs
- **Les 4 sorties de l'Arduino forment un nombre en binaire**
  - Le décodeur BCD-to-7-segments allume les segments en accord avec le nombre formé par les 4 entrées
  - Le décodeur SN74LS48 fournit des sorties à 5V pour allumer les segments (GND commun)
    ► une sortie à 1 allume son segment
  - Le décodeur SN74LS47 fournit des sorties à 0V pour « tirer » à zéro volt les sorties des segments (+5V commun)
    ► sortie à 0 allume le segment (souvent plus robuste)
  - Une résistance de 300Ω pour chaque segment permet de ne pas détruire le décodeur et/ou l'afficheur

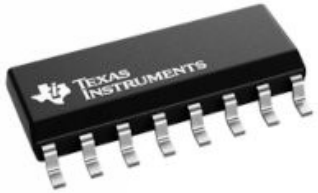


BCD – 7 segments

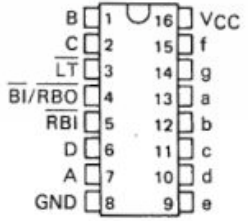
- Extrait de la datasheet SN74LS47
  - VCC : 5V
  - Si on « tire » les broches de l'afficheur 7 segments à GND pour les allumer, il faut un afficheur 7 segments à anode commune (+ commun)
- Penser à mettre des résistances de 330Ω en série avec chaque LED de segment

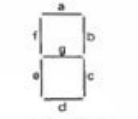






(TOP VIEW)





SEGMENT IDENTIFICATION

'46A, '47A, 'LS47 FUNCTION TABLE (T1)

DECIMAL OR FUNCTION	INPUTS						BI/RBO†	OUTPUTS							NOTE
	LT	RBI	D	C	B	A		a	b	c	d	e	f	g	
0	H	H	L	L	L	L	H	ON	ON	ON	ON	ON	ON	OFF	1
1	H	X	L	L	L	H	H	OFF	ON	ON	OFF	OFF	OFF	OFF	
2	H	X	L	L	H	L	H	ON	ON	OFF	ON	ON	OFF	ON	
3	H	X	L	L	H	H	H	ON	ON	ON	ON	OFF	OFF	ON	
4	H	X	L	H	L	L	H	OFF	ON	ON	OFF	OFF	ON	ON	
5	H	X	L	H	L	H	H	ON	OFF	ON	ON	OFF	ON	ON	
6	H	X	L	H	H	L	H	OFF	OFF	ON	ON	ON	ON	ON	
7	H	X	L	H	H	H	H	ON	ON	ON	OFF	OFF	OFF	OFF	
8	H	X	H	L	L	L	H	ON	ON	ON	ON	ON	ON	ON	
9	H	X	H	L	L	H	H	ON	ON	ON	OFF	OFF	ON	ON	
10	H	X	H	L	H	L	H	OFF	OFF	OFF	ON	ON	OFF	ON	
11	H	X	H	L	H	H	H	OFF	OFF	ON	ON	OFF	OFF	ON	
12	H	X	H	H	L	L	H	OFF	ON	OFF	OFF	OFF	ON	ON	
13	H	X	H	H	L	H	H	ON	OFF	OFF	ON	OFF	ON	ON	
14	H	X	H	H	H	L	H	OFF	OFF	OFF	ON	ON	ON	ON	
15	H	X	H	H	H	H	H	OFF	OFF	OFF	OFF	OFF	OFF	OFF	
BI	X	X	X	X	X	X	L	OFF	OFF	OFF	OFF	OFF	OFF	OFF	2
RBI	H	L	L	L	L	L	L	OFF	OFF	OFF	OFF	OFF	OFF	OFF	3
LT	L	X	X	X	X	X	H	ON	ON	ON	ON	ON	ON	ON	4