Initiation à Arduino

Préambule

Ce tutoriel a pour origine un diaporama utilisé dans le cadre d'un atelier d'initiation à Arduino mis en place au KonkArLab et ouvert à tous. L'atelier comprend 4 séances pratiques faisant suite à une séance de présentation générale de l'environnement d'Arduino.
Après la présentation générale, le tutoriel décrit, sur deux chapitres, les principaux ordres du langage de programmation. Les deux chapitres suivants traitent du montage de composants usuels et sont accompagnés de divers exercices. Enfin, des informations pratiques sont fournies en complément.
Il est conseillé de consulter ce tutoriel dans l'ordre de présentation, chaque chapitre pouvant présenter des exercices s'appuyant sur des éléments vus dans un chapitre précédent.

Charles

Version 2 - 01/2023

Sommaire

** l'astérisque indique la présence d'exercice.*

INITIATION
A
ARDUINO

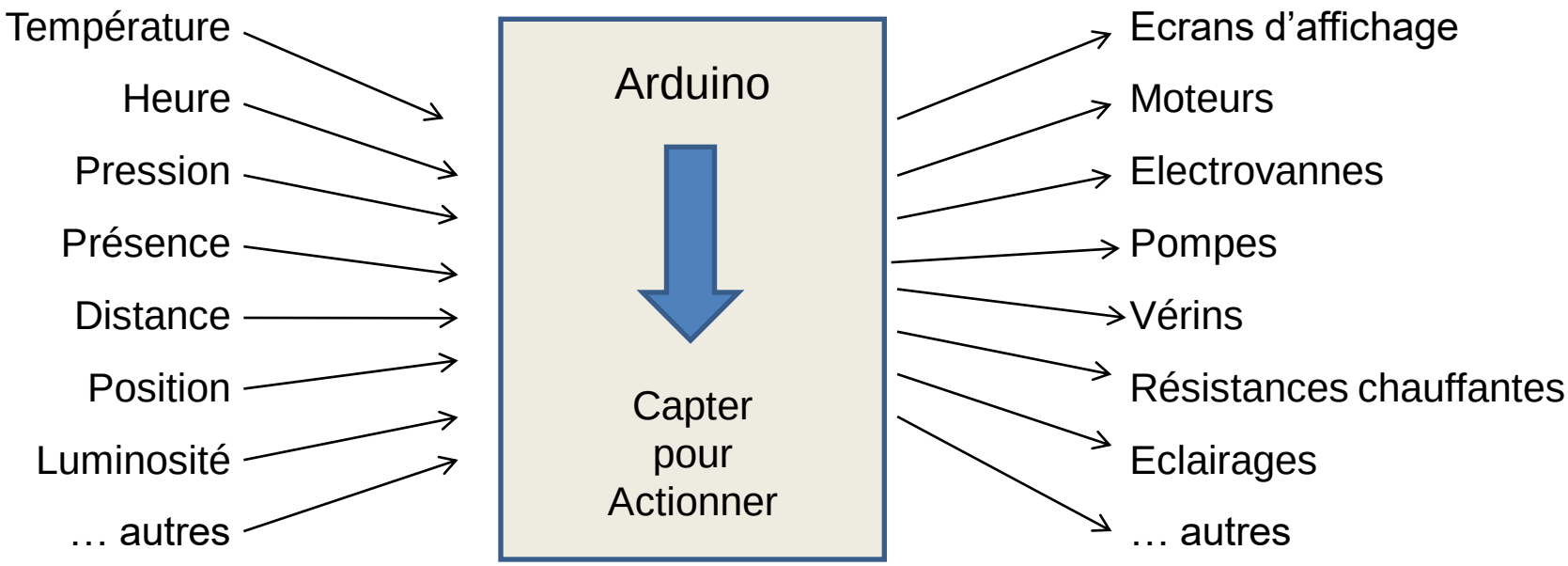
L’univers Arduino :

Un aperçu de la plateforme Arduino et de ses cartes d’interface

Konk Ar Lab
Le FabLab de la Baie

L'univers Arduino

Un **microcontrôleur** qui agit
en interaction avec son environnement



(et aussi le moniteur-série… en entrée et en sortie)

Le microcontrôleur est piloté par un programme (sketch), soit fourni avec les composants, soit rédigé ou complété par l'utilisateur.

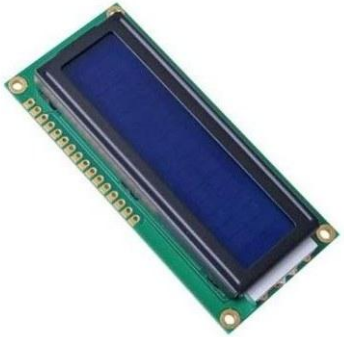
Des exemples de composants, capteurs et actionneurs . . . .

PhotoRésistance

Capteur de température

Capteur d'inclinaison

Bouton poussoir



Ecran LCD

LED

Buzzer

Servo-moteur

Autres exemples de capteurs . . . .

Potentiomètre

Capteur de son

Capteur de gaz

Capteur d'empreinte digitale

Joystick

Caméra infrarouge

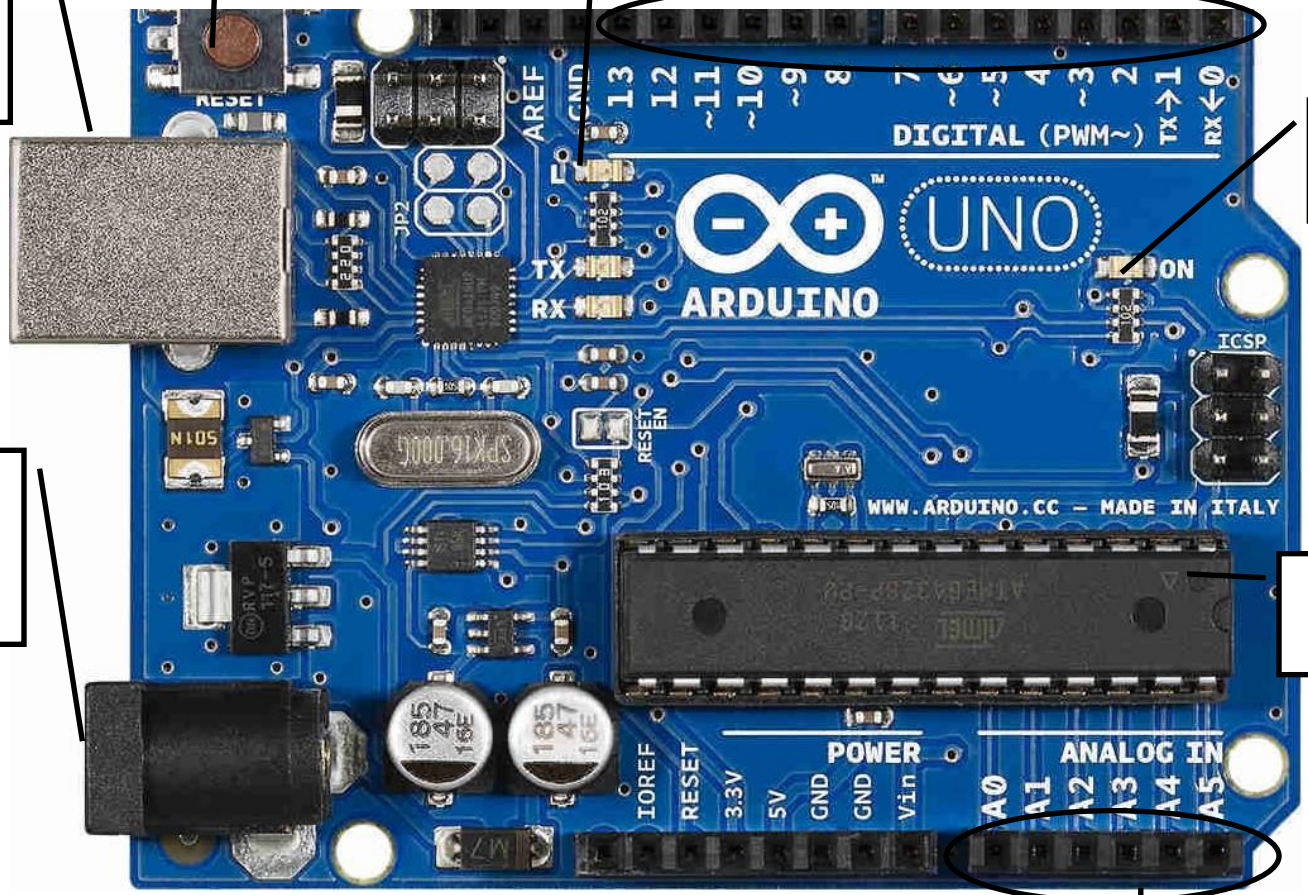
Présentation de la carte Arduino Uno*

Prise reliée à un port USB de l'ordinateur pour l'alimentation de la carte et pour **téléverser** les programmes

Bouton RESET

LED de test connectée à **D13**

14 entrées/sorties **numériques** D0 à D13



LED témoin alimentation

Prise pour autre alimentation externe

Microcontrôleur ATmega328

6 entrées **analogiques** A0 à A5

** il existe de nombreux modèles de cartes Arduino…*

Les broches de branchement des composants

1) Entrées/sorties numériques (digitales) D0 à D13

Chacune des broches D0 à D13 peut être configurée par programmation en entrée ou en sortie.

Les signaux véhiculés par ces broches sont des signaux **logiques** car ils ne peuvent prendre que **deux** états : HAUT (5 Volts) ou BAS (0 Volt). Ils ne peuvent pas fournir ou absorber un courant supérieur à **40 mA** environ (200 mA pour l'ensemble des connecteurs).
Cependant, les broches repérées par un **~** (tilde) sont dites sorties **PWM** (Pulse Width Modulation ou Modulation à largeur d'impulsion) car elles possèdent des propriétés de broches **analogiques** (valeurs admises de 0 à 255).
En pratique, les broches D0 et D1 sont réservées pour la liaison série asynchrone (port COM).

2) Entrées analogiques A0 à A5

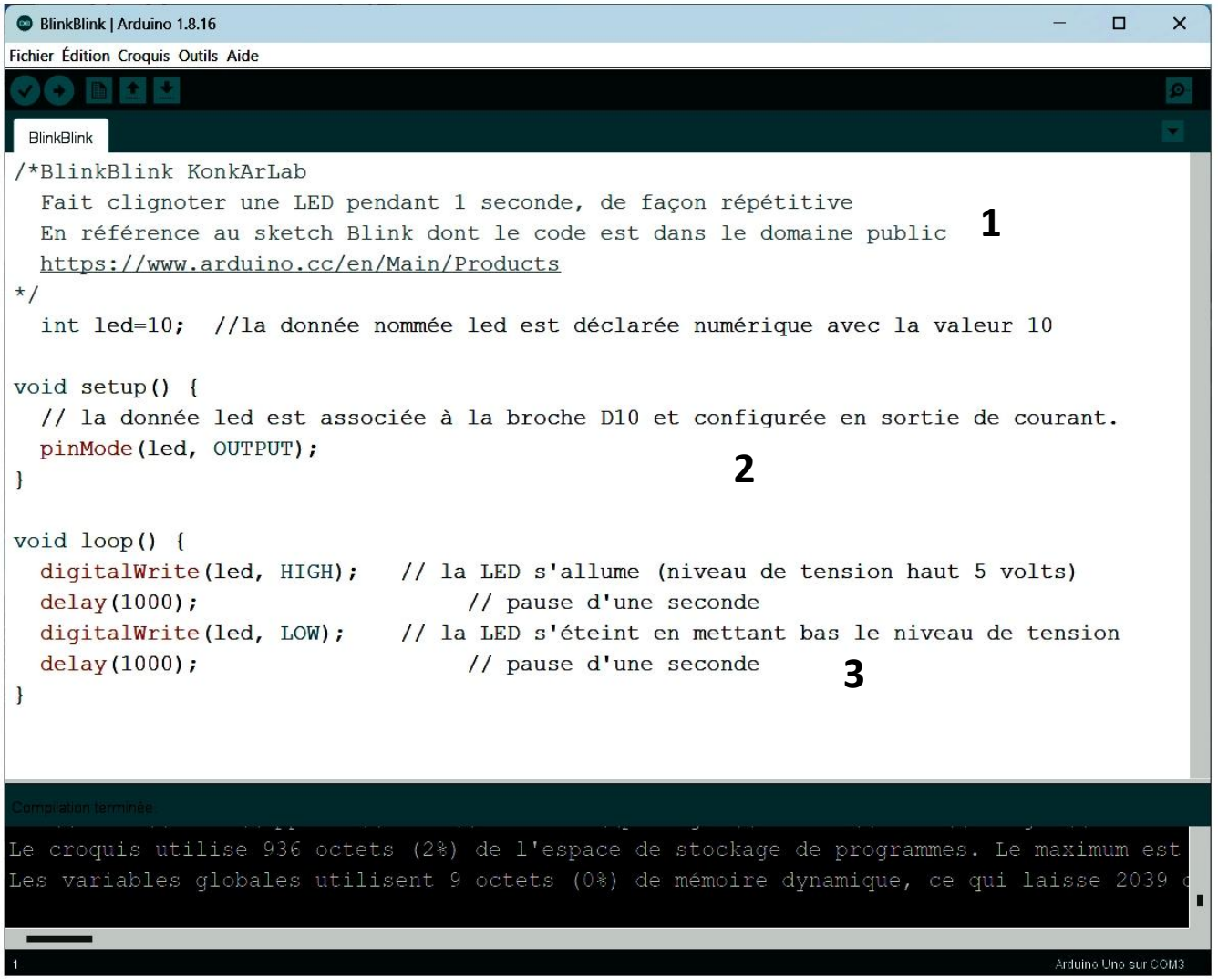
Ces six **entrées** analogiques sont prises en charge par un CAN (Convertisseur Analogique Numérique) dont la fonction est de convertir toute tension analogique comprise entre 0 et 5 Volts en une valeur numérique comprise entre 0 et 1023.

Commande de l'E.D.I. et affichage d'un programme

Commandes de l'E.D.I.
(l'Environnement de Développement Intégré de l'Arduino)

Un programme est composé de 3 parties

Le programme **BlinkBlink** fait clignoter une LED.
Il fait partie des exercices du tutoriel.



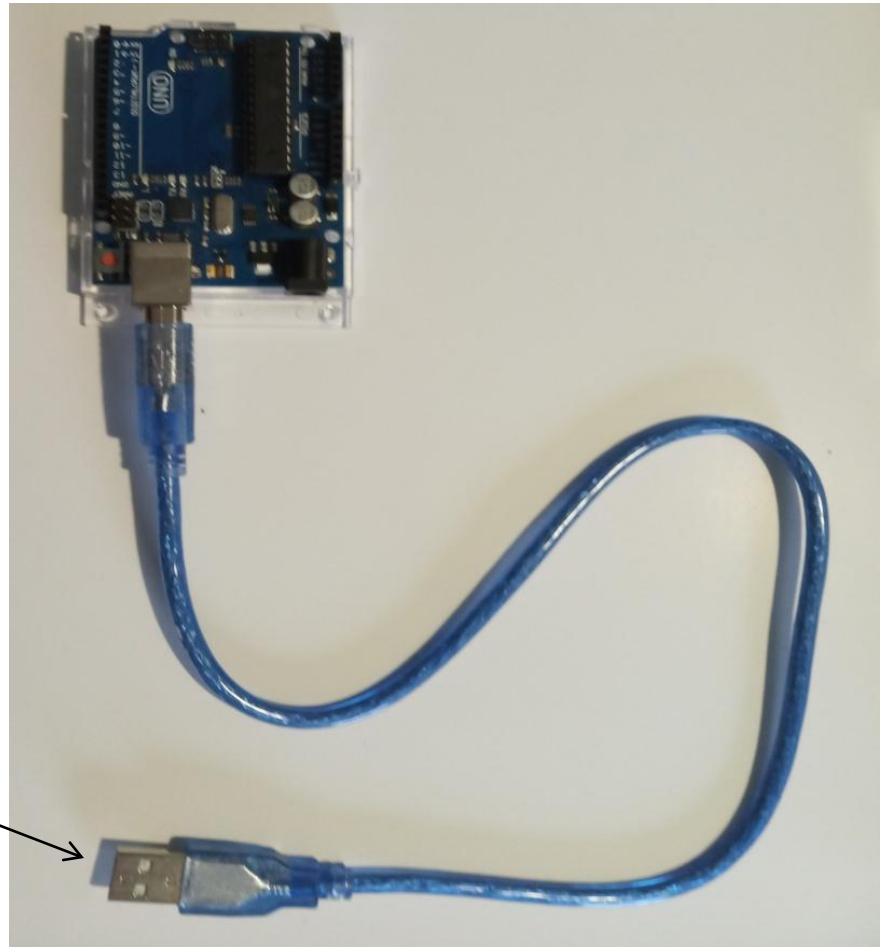
La programmation de base : tableau de l'ensemble des ordres pour la gestion des entrées/sorties de la carte Arduino

Ordres/fonctions	Description générale
pinMode(pin,mode)	Configuration d'une broche **numérique** en entrée ou en sortie (mode INPUT ou OUTPUT). **=> Exemple** : `pinMode(7,INPUT);`
digitalWrite(pin,valeur)	Pour une broche numérique configurée en sortie par `pinMode()` cette fonction permet de mettre sa tension à 5 v pour la valeur HIGH (ou 3,3 v sur les cartes 3,3 v) ou à 0 v (masse) pour la valeur LOW.
digitalRead(pin)	Lecture de la valeur de tension de la broche numérique configurée en entrée, soit HIGH ou LOW. **=> Exemple** : `digitalRead(boutonPoussoir);`
analogWrite(pin,valeur)	Écrit une valeur analogique sur une broche numérique marquée PWM. Valeur : le cycle de service est entre 0 (toujours éteint) et 255 (toujours allumé). Peut être utilisé pour allumer une LED à différentes intensités ou conduire un moteur à différentes vitesses.
analogRead(pin)	Lecture de la valeur de la tension présente sur une entrée analogique (broches A0 à A5 sur la Uno et broches numériques marquées PWM). Cette valeur peut être comprise entre 0 et 1023 pour A0 à A5.

Pin = numéro de broche ou nom de variable.

Les ordres du tableau permettent l'accès à tous les composants !

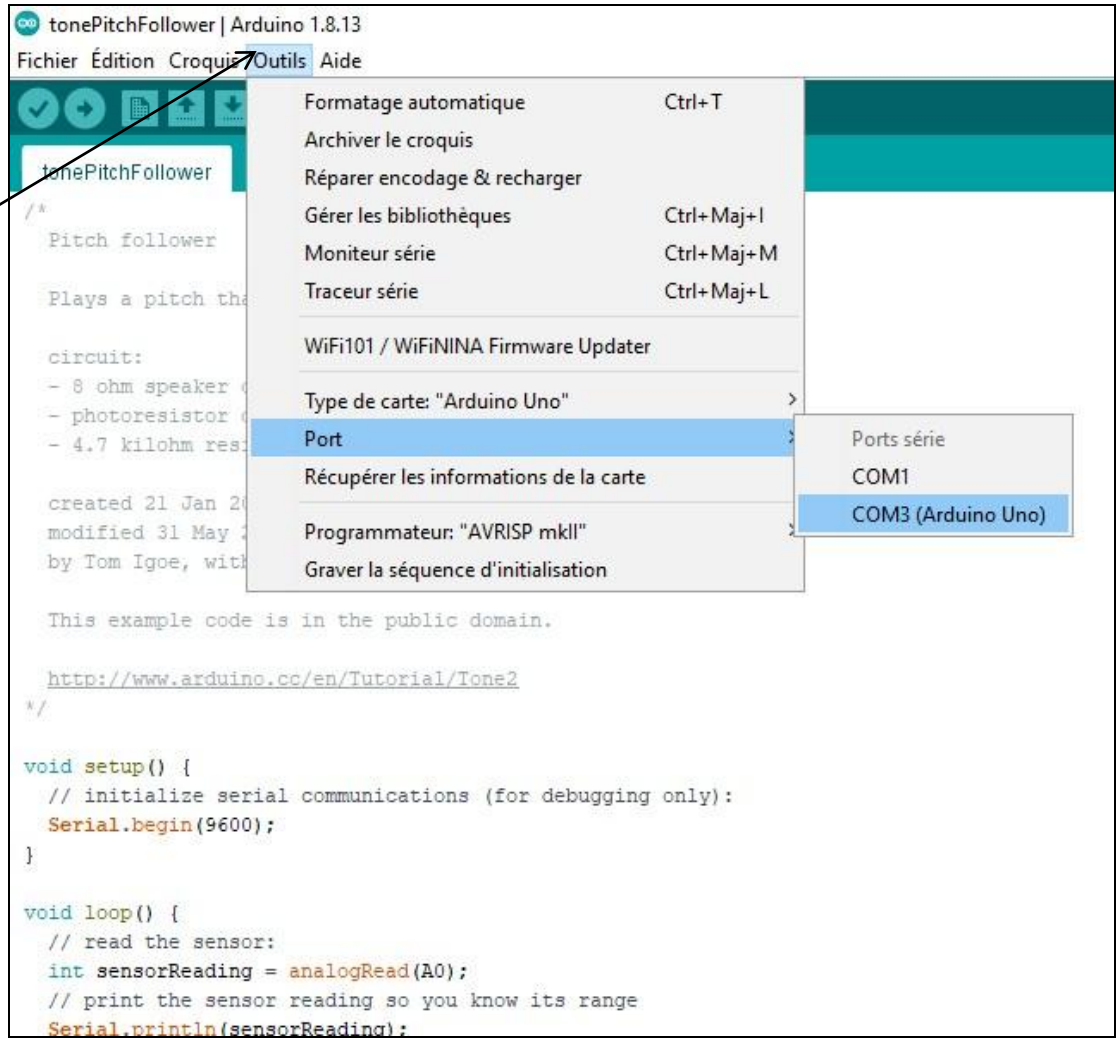
Pour télécharger le programme dans la carte, il faut relier la carte à l'ordinateur via un câble USB type A/B



Prise USB reliée à l'ordinateur

Il convient ensuite d'activer le bon port du PC pour assurer la connexion

Utilisation de la commande **Outils.**

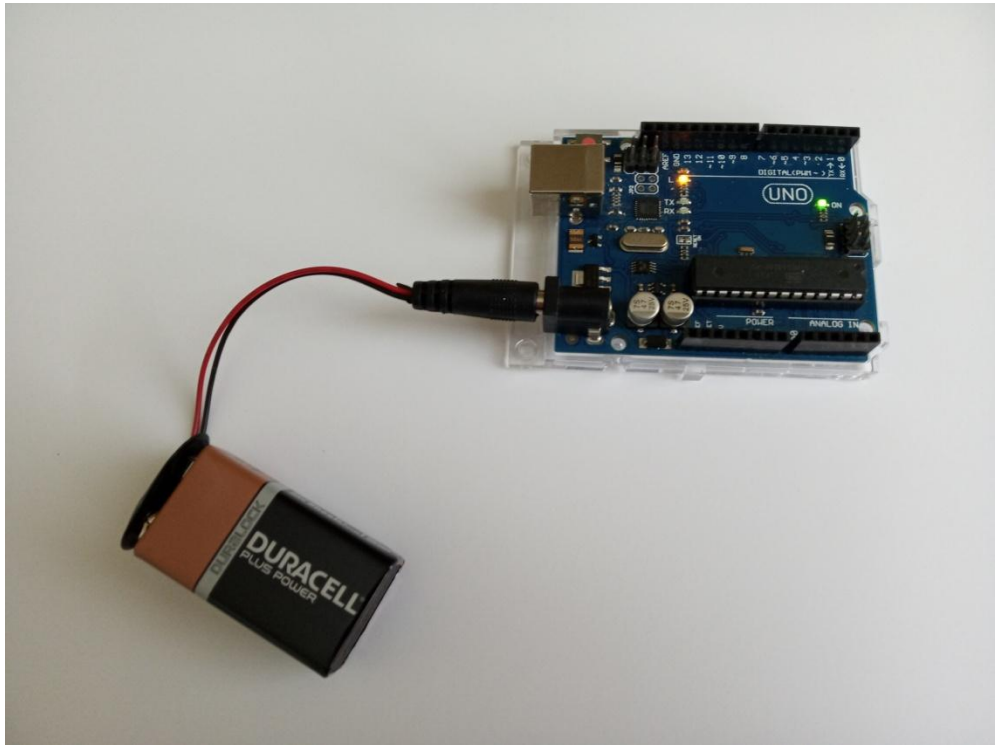


Port connecté

Autonomie de la carte Arduino

La carte peut être alimentée par un bloc secteur (tension entre 7 et 12 Volts, 1 Ampère). Comme elle consomme très peu (0,5 W), elle peut également être alimentée par une simple pile 9 V. Elle comporte un régulateur de tension à 5 volts.

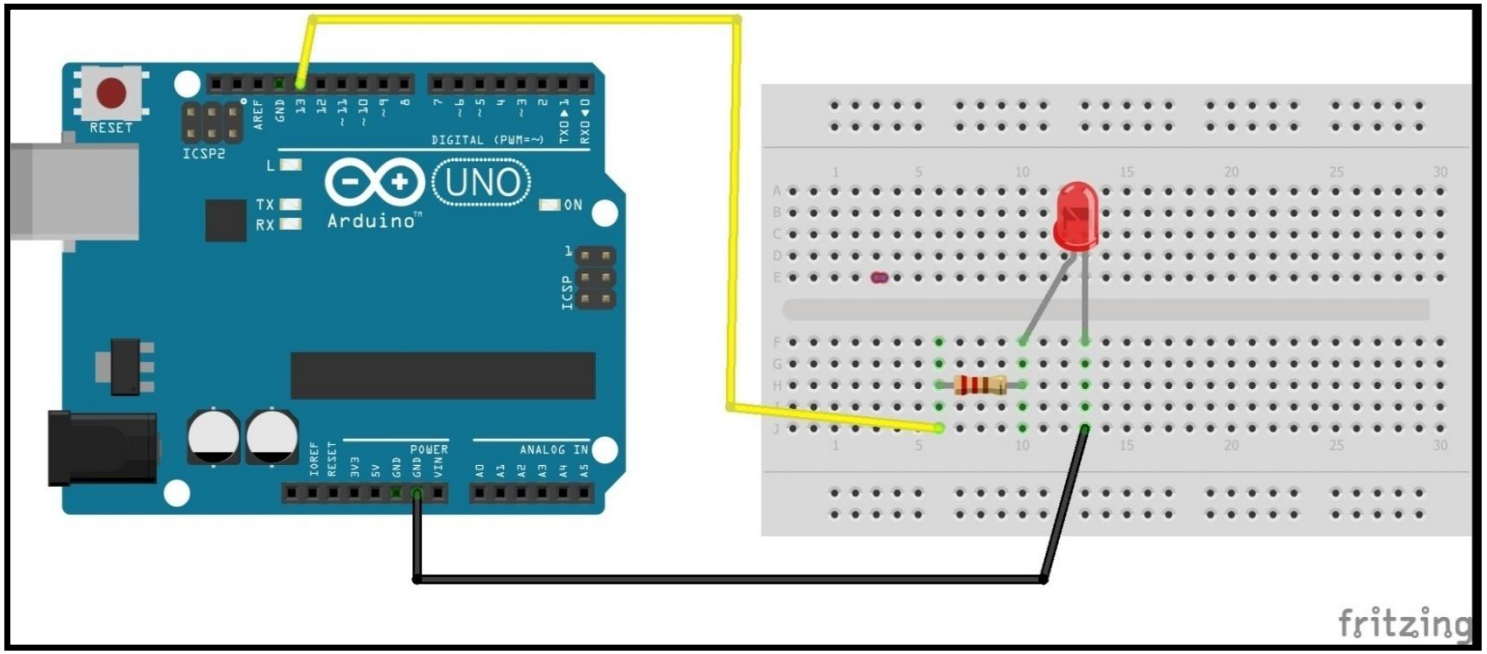
Exemple de carte alimentée par une pile 9 V.



La carte **fournit** des tensions stabilisées à 3,3 ou 5 Volts pour l'alimentation des périphériques (actionneurs).

Le complément à Arduino : la plaque d'essai (breadBoard)

Elle permet de câbler de nombreux composants sans nécessiter de soudure et de garder le montage manuel modifiable et entièrement démontable.



Le montage sur cette plaque d'essai (ici le petit modèle) montre une LED branchée sur la broche D13 de l'Arduino avec une résistance placée en série.

Le schéma a été dessiné avec le logiciel **Fritzing,**

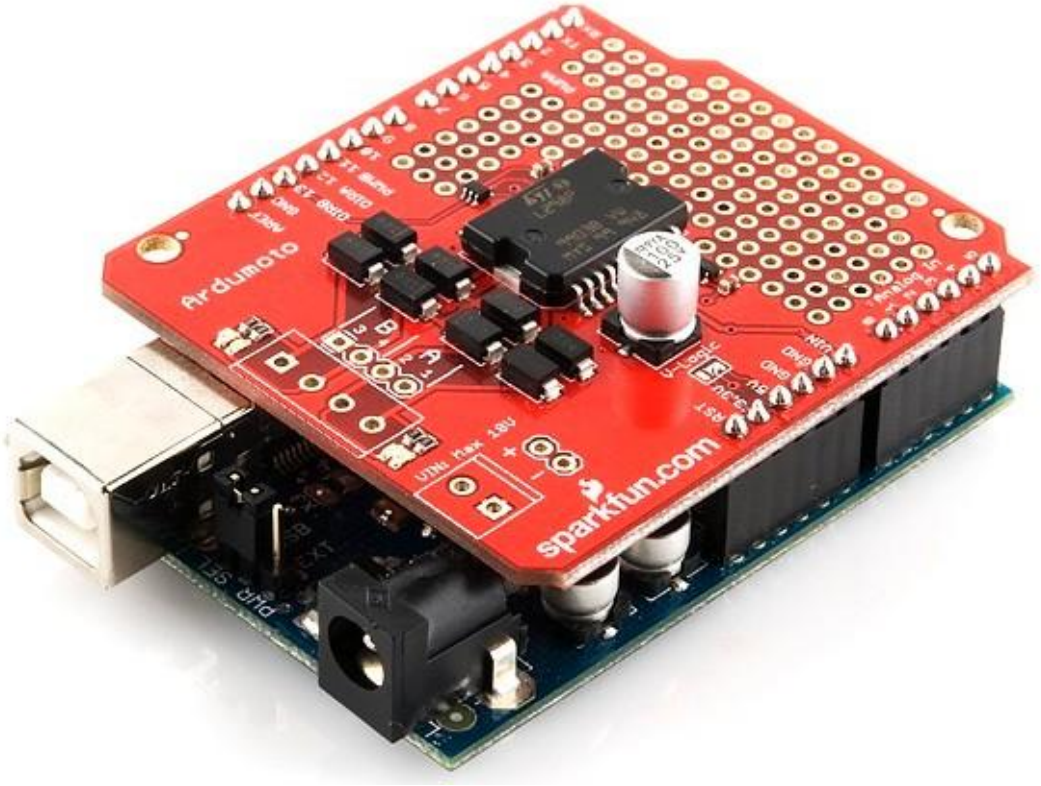
Des exemples de composants d'interface (shields) . . . .

Carte pour l'affichage

Ce *shield, en plus de son écran LCD (deux lignes de 16 caractères blancs à rétroéclairage bleu), comporte six boutons-poussoirs pour commander l'affichage de différents messages* (select, left, down, up, right et reset*). Il utilise les broches D4 à D10 pour l'afficheur et la broche A0 pour les boutons-poussoirs.*
L'ajustement de la luminosité peut être effectué via un potentiomètre.

La bibliothèque LiquidCrystal fournit les fonctions de commande de l'affichage. Elle est disponible avec la bibliothèque de l'Arduino.

Carte pour la commande de moteurs



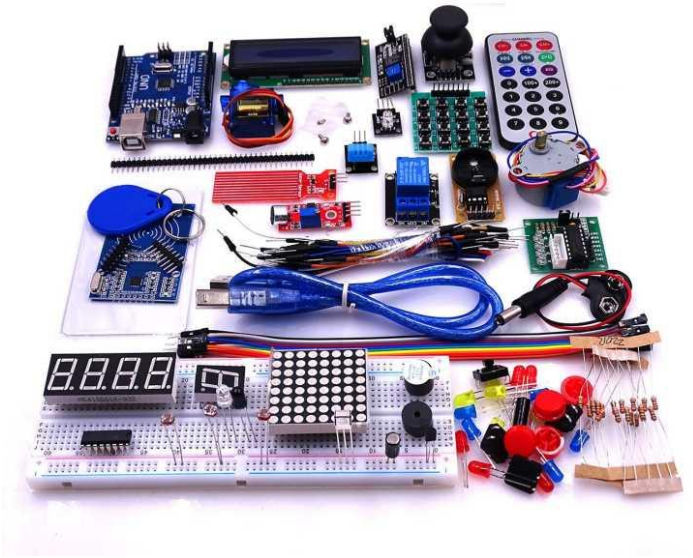
Ce *shield permet de piloter jusqu'à deux moteurs électriques simultanément (idéal pour un petit robot roulant à deux roues motorisées indépendantes).*

Avec seulement deux connecteurs numériques utilisés par moteur on peut contrôler le sens et la vitesse de rotation des moteurs.

Carte pour communiquer

Pour rendre disponible l'état d'un capteur de température sur un réseau local Ethernet ou sur Internet, ce *shield évolué avec une bibliothèque très complète et de nombreux exemples de programmes, saura transformer l'Arduino en véritable serveur web.*

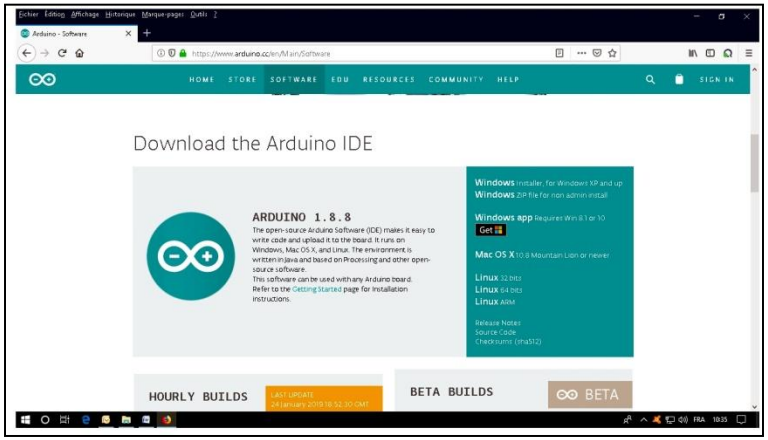
Le coffret proposé par Konk Ar Lab



Contenu indicatif (peut varier) :
Câble USB – câbles de saut/jumpers – platine de prototypage – lumières LED (15) – résistances – potentiomètre – ligne dupond femelle à mâle – buzzer – récepteur Infrarouge – capteur de température LM35 – capteur de flamme – Interrupteurs à bille 520D (2) – photorésistances (2) – boutons poussoir (4) – caches-boutons (4) – télécommande – registre à décalage/shift register 1X74HC595 – afficheur 4 chiffres – module matriciel 8x8 – afficheur 1 chiffre – carte de pilote de moteur pas à pas – moteur pas à pas – servo 9G – écran LCD 1602 HC – module de joystick XY – capteur de température et d'humidité – capteur de niveau d'eau – module RFID – porte-clés RFID – carte blanche RFID – module sonore – module relai – module d'horloge – module de couleur rvb – batterie 9V.

Installation de l'environnement de développement Arduino/Genuino (1/2)

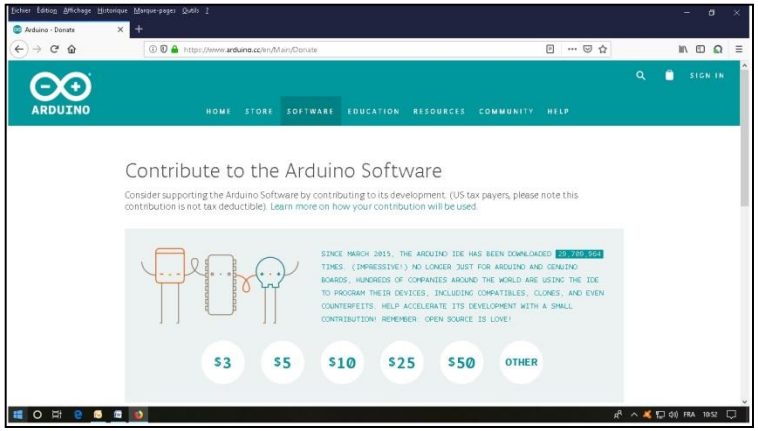
Pour télécharger l'environnement de développement Arduino **E.D.I.** il faut se rendre sur le site officiel de l'Arduino : https://www.arduino.cc
puis cliquer sur l'onglet *Software/Downloads*.
Sur la page de téléchargement, on choisira la version appropriée :



Lors du clic sur la version choisie, une page faisant appel au don est affichée :

La contribution financière est facultative.
A défaut, on choisit l'option "Just download".

Le téléchargement du logiciel Arduino est gratuit.



Installation de l'environnement de développement Arduino/Genuino (2/2)

Avec la version zip de Windows, on pourra procéder à l'extraction à l'aide de 7-Zip ou WinZip.

Pour Linux, une version zip est également proposée.
La décompression de l'Arduino peut se faire dans un répertoire créé sous la racine et que l'on nomme « Arduino". Par un clic droit sur le fichier **install.sh** on sélectionne l'option **Run in Terminal** ce qui exécute l'installation.
Dans le cas où l'option n'existe pas, on ouvre un terminal. On se rend dans le répertoire "arduino" et on tape la commande ./install.sh ce qui exécute l'installation.
Dans tous les cas, on obtient une icône d'Arduino sur le Bureau.

Après l'implantation, on crée le dossier *Arduino* qui contiendra les programmes (aussi appelés sketches ou croquis) créés par l'utilisateur.
Le lancement d'Arduino affiche l'écran de l'I.D.E. (voir page n° 9).

- La commande *Fichier/Preferences* désigne l'emplacement du carnet de croquis (le dossier *Arduino*).
- La commande *Outils* permet de spécifier le type de carte, notamment Arduino Uno, et le port de liaison avec l'ordinateur.
- La commande *Aide/Reference* fournit de l'aide sur les ordres ou fonctions du langage de programmation de l'Arduino.

Avertissement : dans la suite du tutoriel, on utilise indifféremment les termes de programme, sketch ou croquis pour désigner la même entité.

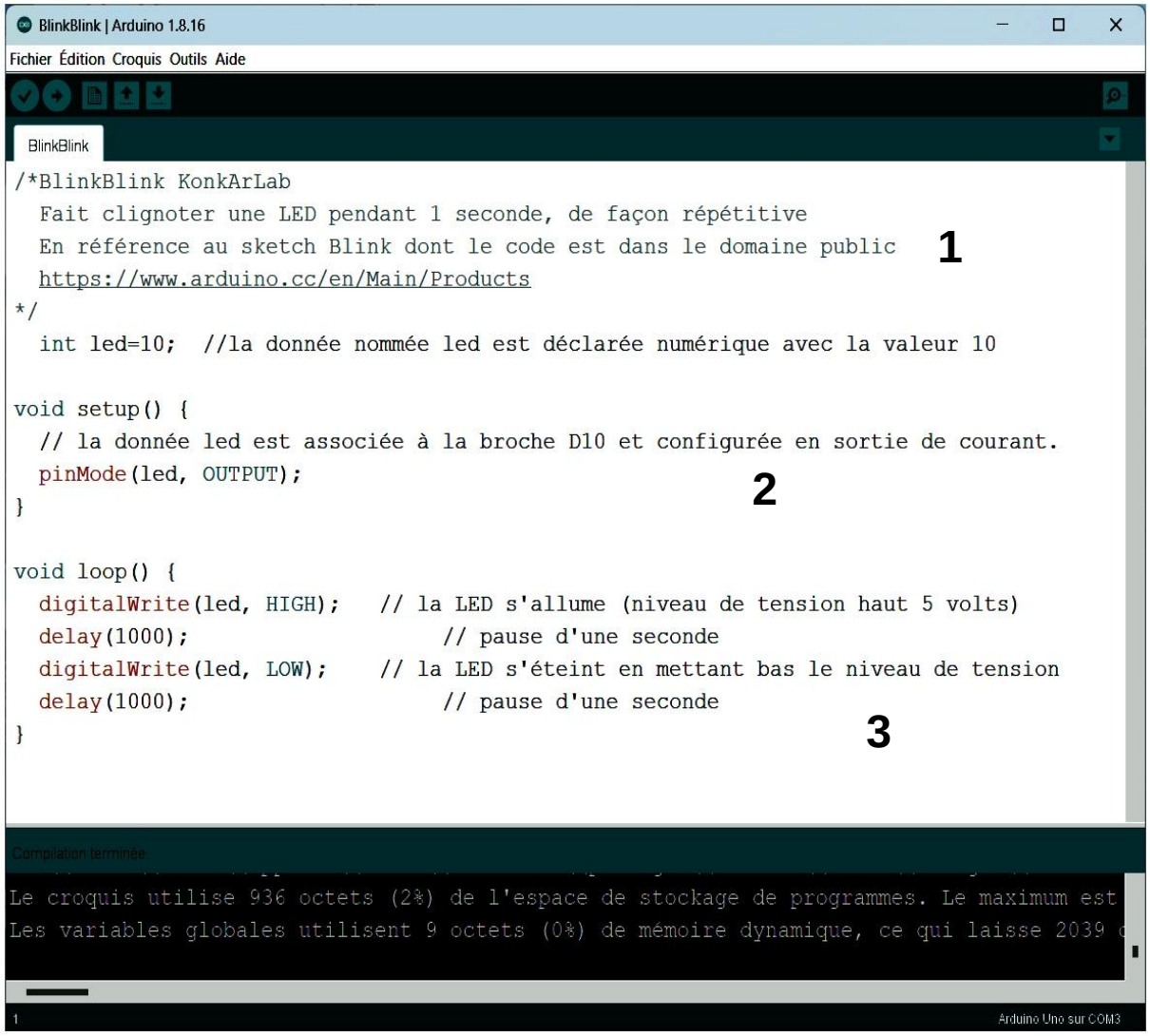
Retour sommaire

INITIATION
A
ARDUINO

La programmation (partie 1/2) :

Le programme et l'E.D.I. – les règles de codage – les types de données – les opérateurs arithmétiques

Structure d'un programme ou sketch



1. L'en-tête (contenu) :

Nom du programme et description de son objet (commentaires).
Déclaration des données constantes et des données variables dites **globales** car elles sont reconnues et utilisables dans tout le programme.

2. Le void setup() {….}

Exécuté **une seule fois**, au lancement du programme.
Pour configurer les entrées/sorties et initialiser des données variables dites locales.

3. Le void loop() {…..}

Boucle qui s'exécute **en continu**.
Les ordres sont exécutés dans la séquence, jusqu'à l'appui sur le bouton **Reset** de la carte.

zone d'information de l'Arduino

Les commandes de gestion de l'E.D.I.

Boutons : Vérifier - Téléverser - Nouveau - Ouvrir – Enregistrer

Exemple : étapes de réalisation d'un programme :

Etape	Action	Résultat
1	Commande **Fichier/Nouveau**	Affichage par l'E.D.I. d'un squelette de programme. Nom : *sketch_novxx*
2	Bouton **Enregistrer** et attribuer un nom au programme	Enregistrement du programme dans le répertoire Arduino. Message *Enregistrement terminé*. Le nom donné au programme est affiché.
3	Ecriture du programme, puis bouton **Vérifier**	Vérification du codage par l'E.D.I. Erreurs signalées à corriger.
5	Correction des erreurs éventuelles. Bouton **Vérifier**	Message *Compilation terminée*. Le codage est conforme. Les tests de bon fonctionnement du programme peuvent s'effectuer.
6	Bouton **Téléverser**	Message *Téléversement terminé*. Puis le programme s'exécute.

Chargement* d'un programme déjà sauvegardé :

1	Commande **Fichier/Ouvrir**	Affichage de la liste des programmes enregistrés dans le répertoire Arduino.
2	Sélection du programme	Affichage du contenu du programme
3	Bouton **Enregistrer** (si modifications du programme)	Message *Enregistrement terminé*.

* Le chargement peut aussi se commander en sélectionnant le programme dans le dossier « Arduino » du PC (avec la possibilité de suppression).

Exercice

Exercice chapitre n° 2 : Programme Blink = Faire clignoter la LED interne n° 13 de la carte Arduino

a) Charger le programme **Blink** de la bibliothèque d'Arduino : commande *Fichier/Exemples/01.Basics.*
Objet du sketch : clignotement de la LED13 de la carte.

LED_BUILTIN désigne la LED de test montée en interne sur la carte et fixée à la broche 13. C'est un mot réservé d'Arduino. Il n'a donc pas besoin d'être déclaré dans le programme.

La fonction **delay()** permet de marquer une pause entre les instructions.
Son paramètre donne la durée de la pause en nombre de millisecondes (1 s = 1000 ms).

La fonction delay() est nécessaire pour voir le clignotement, car il faut ralentir la cadence des « Loop » de l'Arduino.
En effet, le processeur est cadencé à 16 MHz = 16 000 000 d'actions machine par seconde.
Dans le cas de notre programme, il clignote à un peu moins de 77 KHz, soit 77 000 fois j'allume/j'éteins par seconde (chaque fonction "allume/éteint" est elle-même composée de plusieurs actions machine !). Or notre œil ne perçoit des images différentes qu'en dessous de 25 Hz.

b) Actionner le bouton *Vérifier.* Une erreur de compilation peut être signalée si l'Arduino n'a pas été lancé en tant qu'Administrateur.
ATTENTION : la commande propose parfois d'enregistrer le sketch. Ne pas le faire ici à ce stade.
Pour exécuter le programme, on branche une carte Arduino.

Bouton *Téléverser* pour enregistrer le programme sur la carte. La LED de test clignote.
Modifier les valeurs des delay() : on obtient des clignotements plus ou moins rapides. Faire des essais.

c) **Commande** *Fichier/Enregistrer sous* pour le stockage du sketch. On nommera le programme **BlinkBlink**. Il sera utilisé dans le chapitre suivant. On peut aussi supprimer les commentaires inutiles.

Les règles de codage

Règles de syntaxe :

Des noms symboliques sont attribués aux **données** pour mieux les identifier lors de leur utilisation dans le programme. On distingue 2 natures de données :

Les constantes : DUREE, VALEURMAXI, PLAFOND

Les variables : nbBilles, tempsDePause, temperatureAir

Les noms ne doivent pas comporter de caractère accentué, ni d'espace, ni commencer par un chiffre.

Une **donnée variable** définie dans un bloc de code (entre des accolades) ne peut être exploitée que dans ce bloc. Pour pouvoir être utilisable dans tout le programme, elle devra avoir été définie dans la partie en-tête du programme (avant le setup).

Ponctuation :

Toute **ligne** de code se termine par un point-virgule « **; »**
Le contenu d'une **fonction,** exemple du void setup(), est délimité par des accolades : **{ }**
Les **paramètres** d'une fonction sont contenus entre parenthèses : **( )**

Commentaires (utilisés pour documenter le programme) :

Contenus sur une seule ligne : ***//*** commentaires
Contenus sur plusieurs lignes : ***/****première ligne,…. lignes suivantes…, ****/***dernière ligne

L'indentation *s'obtient par la commande* ***Outils/Formatage automatique*** *(et la commande* ***Edition*** *pour augmenter/réduire).*

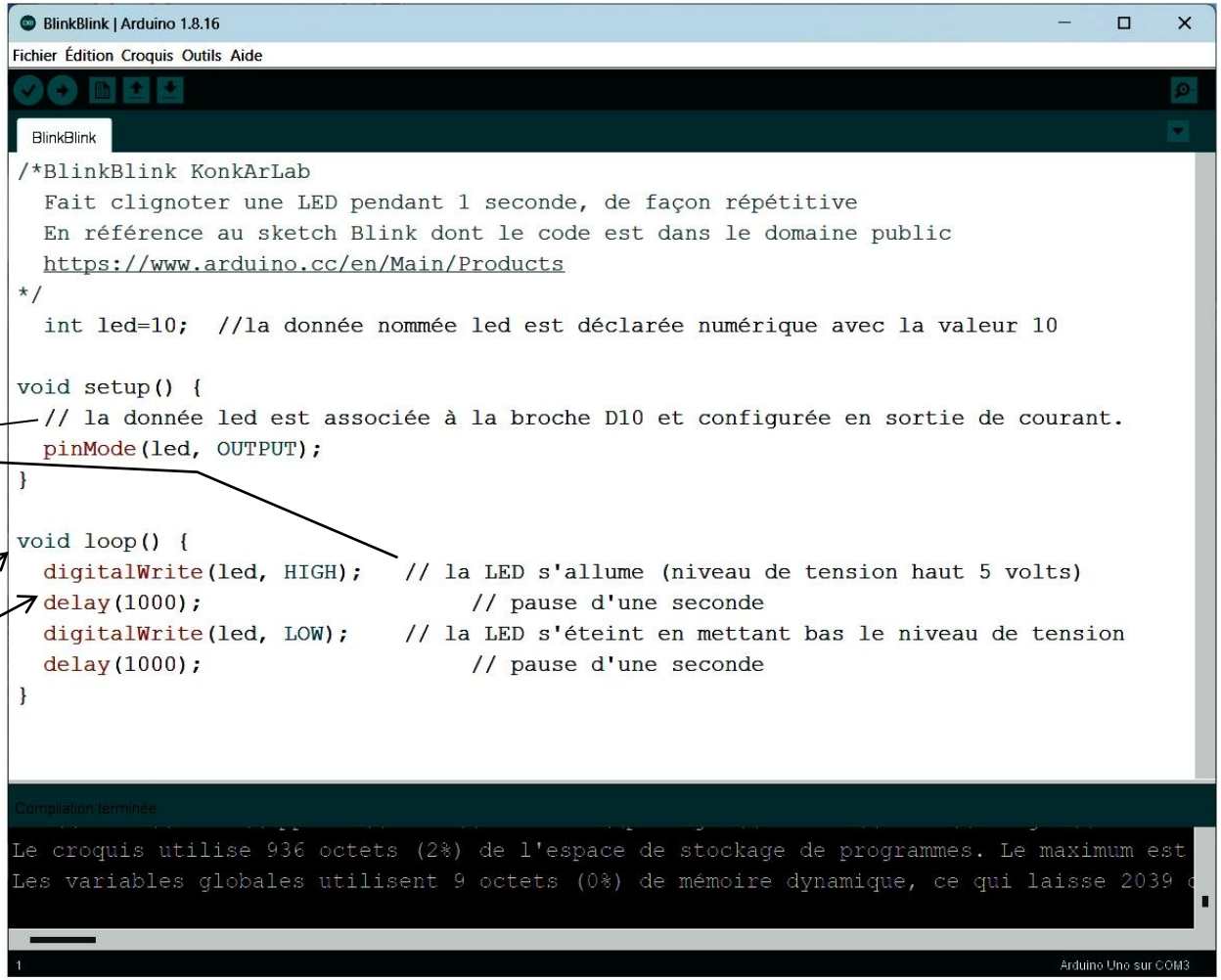
Exemple de commentaires

Emploi de balises :

/* ….*/
sur
plusieurs
lignes

//
sur une
seule ligne

et
l'indentation



Un programme est identifié par un nom. Il est composé de données, d'ordres et de fonctions.

Les types de données et l'assignation des valeurs

Les données sont associées à un type (taille en octets) selon leur rôle :

Type	Taille	Contenu possible
boolean	1	deux valeurs logiques : true/false (Oui/Non)
char	1	un caractère ou un entier entre -128 et +127
unsigned char	1	un caractère ou un entier entre 0 et 255
byte	1	un entier entre 0 et 255
int	2	un entier entre -32 768 et 32 767
unsigned int	2	un entier entre 0 et 65 535
float	4	un décimal à 7 chiffres après la virgule
long	2	un entier entre -2 147 483 648 et 2 147 483 647
string	n	une suite de caractères

Exemples d'assignation de valeurs aux données :

Constantes :

const type NOM=valeur ; Exemple : `const int CONNEXION=13;`

Variables :

```
Int tpsDePause;
tpsDePause=100;
```

ou `int tpsDePause=100;`

Déclaration multiple :

```
int led1, led2, led3;
```

Les opérateurs arithmétiques* associés aux données

Opérateurs de base

= égalité
+ addition
- soustraction
* multiplication
/ division

et les nombres au hasard : `random(n,p)`

Exemple :
```
resultat = (val1 - val2)*20.6;
```

Opérateurs composés

++ incrémentation
-- décrémentation
+= addition composée

Remarque :
Trois possibilités pour incrémenter une variable :
```
variable = variable + 1
variable+=1
variable++
```

** Les opérateurs dits de comparaison sont vus dans le prochain chapitre.*

Retour sommaire

INITIATION
A
ARDUINO

La programmation (partie 2/2) :

Structures de contrôle et opérateurs de comparaison – Plaque d'essai – Led, résistance et loi d'Ohm

Les structures de contrôle : conditions appliquées à des ordres du programme pour conditionner leur exécution

if (…) { ….} else { … }	**si** la condition est réalisée, exécution des ordres1
	sinon
	exécution des ordres2
	(le sinon est facultatif)
		`if (allumerLed == 1) {`
		`  digitalWrite(ledPin, HIGH);`
		`}`
		`else {`
		`  digitalWrite(ledPin,LOW);}`

while	Tant que la condition se réalise, exécuter les ordres donnés dans la boucle.	`while(a<5) {`
		`  a++;  //incrémente la variable a`
		`}`

do while	do { ordres à exécuter dans la boucle; } while (expression);	Idem à while mais le code de la boucle est toujours exécuté au moins une fois.

for	Boucle qui, pour telle variable allant de telle valeur à telle autre valeur, selon un pas donné, exécute des ordres.	`for(int t=0;t<10;t++) { //à chaque tour, +1 dans t`
		`//code à exécuter. Il sera exécuté 10 fois.`
		`}`

switch case break	exécute les ordres du cas n selon la valeur de la variable.	`switch(variable) {`
		`case 1:  //cas où la variable contient 1`
		`  digitalWrite(numLed3,HIGH);`
		`  break;  //sortie du cas`
		`case 2:  //cas où la variable contient 2`
		`  digitalWrite(numLed3,LOW);`
		`  break;`
		`default:  //facultatif, cas d'autres valeurs`
		`….`
		`break;`
		`}`

Les opérateurs de comparaison associés aux données

Opérateurs de comparaison

== égal à
!= non égal à
< inférieur à
> supérieur à
<= inférieur ou égal à
>= supérieur ou égal à

Opérateurs booléens

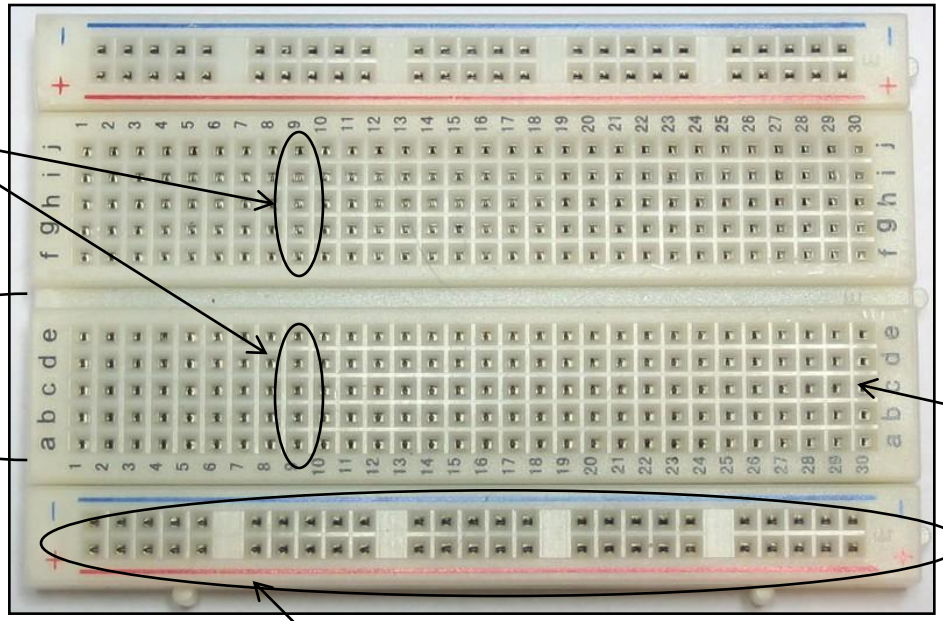
&& ET
|| OU
! NON

Exemple de condition ET. Cas où deux broches de l'Arduino doivent être activées en même temps pour permettre l'exécution des ordres :

```
if (digitalRead(pinLed1) == HIGH && digitalRead(pinLed2) == HIGH) {
// ordres ...
}
```

La plaque d'essai ou breadBoard*

La plaque d'essai ou breadBoard permet de réaliser, sans soudure, un montage électronique en branchant les pattes des composants et les fils de cablage dans les trous.



Les deux parties de la carte sont indépendantes

Les trous d'une colonne sont reliés entre eux

Les trous d'une ligne sont reliés entre eux

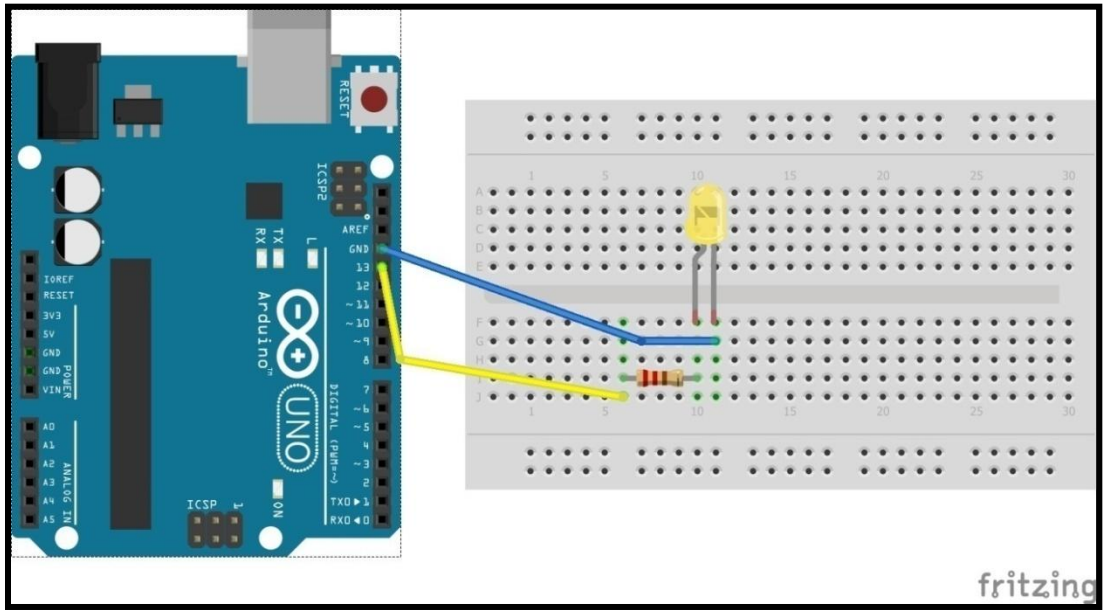
Les trous d'une ligne ne sont pas reliés entre eux

Ces 2 lignes sont dédiées à l'alimentation du circuit (connexion avec le + et avec le GRD de la carte Arduino)

(*) Le mot breadBoard, littéralement « planche à pain », fait l'objet de nombreuses traductions : platine de prototypage ou d'essai, plaque de cablage rapide, plaque d'essai ou de test…

La LED

Schéma de montage d'une LED sur la breadBoard :



La patte la plus longue (pôle +) se place du côté de l'arrivée du courant. Ici, il est fourni par la borne 13 de l'Arduino qui émet du 5 volts par programmation.

L'autre patte (pôle –) est connectée au GND.

Une résistance de 220 Ohms est placée dans le circuit de la LED, avant ou après celle-ci pour la protéger d'un risque de surtension.

Si on suit le parcours du courant, il part de la broche 13, il passe par la résistance, puis par la LED, puis il rejoint le GND (Ground ou Terre) de l'Arduino. La résistance protège la LED d'un risque de survoltage. Pour sa part la LED de test, interne à la carte arduino, est dotée de sa propre résistance.

Exos 1 et 2

Exercice 1 chapitre n° 3 : BlinkBlink avec LED et breadboard

Matériel utilisé : carte Arduino et cable de liaison – plaque d'essai – LED – résistance de 220 Ohms minimum.
Utilisation du sketch **BlinkBlink**, créé au chapitre n° 2, qui fait clignoter la LED de test de la carte.
Objectif :
Modifier le sketch **BlinkBlink** pour faire clignoter une nouvelle LED branchée, elle-aussi, sur la broche 13 de la breadBoard.
Montage :
Il figure sur la page précédente qui présente la LED.

Pour calculer la valeur de la résistance, on utilise la loi d'Ohm : U = R x I ou R = U/I
La tension d'alimentation fournie par l'Arduino est de 5 V et la tension aux bornes de la LED doit être de 1,2 V en fonctionnement normal. D'où la tension Ur qui devra être aux bornes de la résistance :
Ur = 5 − 1,2 = 3,8 V
Valeur de la résistance à introduire : R = Ur/I = 3,8/0,02 = 190 Ω
Il s'agira de la valeur minimum de la résistance à utiliser pour être sûr de ne pas griller la LED.
En pratique, on utilisera une résistance de 220 Ohms ou 330 Ohms.

Programmation :
On charge le programme, commande *Fichier/Carnet de croquis* et sélection de **BlinkBlink**.
Le sketch est présenté dans la page du chapitre n°2 qui traite des commentaires. On a supprimé les commentaires en anglais et placé de nouveaux commentaires.
Pour désigner la nouvelle LED, on lui affecte un nom de donnée (constante ou variable) et un type.
D'où, avant le setup() :

```
int led = 13;   //la donnée est une constante de type entier
```

Dans le setup() et dans le loop() on remplace LED_BUILTIN par led.

Puis bouton *Vérifier,* puis *Téléverser* après correction des erreurs éventuelles…. Ça arrive !
Téléchargement dans la carte.
On constate que la nouvelle LED clignote bien au même rythme que la LED de test.

Exercice 2 chapitre n° 3 : Suppression du clignotement de la LED de test
En exécutant le programme, on remarque que la LED clignote ainsi que la LED de test de la carte. En effet, la LED de test est liée à la broche 13.
Pour ne plus la voir s'allumer, nous brancherons la nouvelle LED sur la broche **10** à la place de la broche 13.

D'où, avant le setup() :

```
int led=10;
```

Puis, bouton *Vérifier* puis *Téléverser*.

La LED de test 13 ne clignote plus.

Intérêt de la variable *: si on change de numéro de broche il n'y a qu'une instruction à modifier.*

A noter que la sortie +5V fournit du courant continuellement. En revanche, les broches de 1 à 13 ne délivrent du courant qu'à partir des commandes initiées par programmation (commande pinMode(n,OUTPUT) pour spécialiser la borne n en sortie et commande digitalWrite(n,HIGH) pour émettre du courant).

Le langage de l'Arduino permet d'exploiter chaque broche numérique en entrée ou en sortie..

Retour sommaire

INITIATION
A
ARDUINO

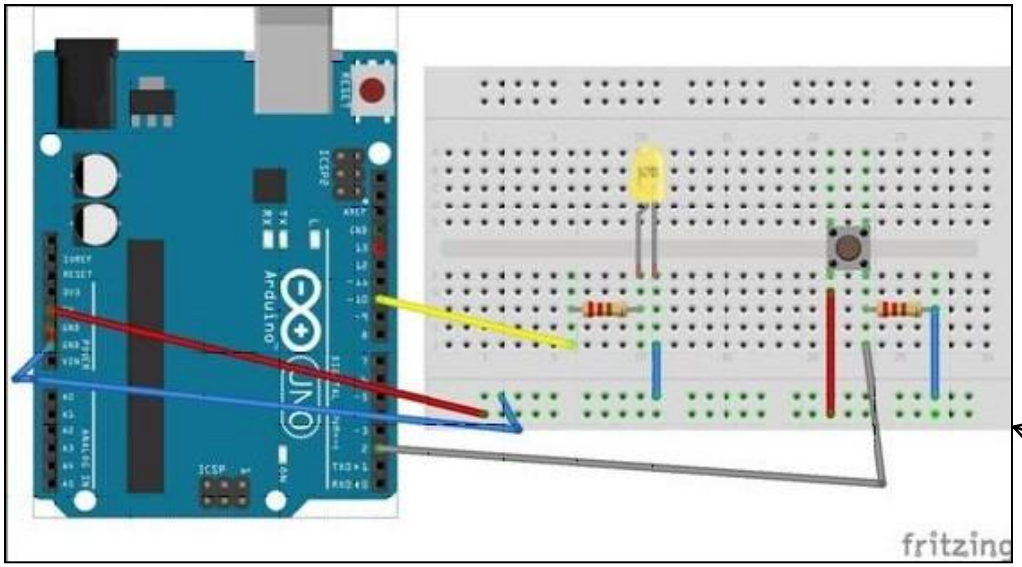
Travaux pratiques (partie 1) :

Le bouton poussoir – le moniteur série en entrée et en sortie

Le bouton poussoir

Un modèle :

Attention, même s'il a 4 pattes, le bouton poussoir est un dipôle. En fait les pattes sont reliées deux par deux. S'il ne fonctionne pas, il faut tourner le bouton d'un quart de tour… ou utiliser les pattes opposées diamétralement. Et contrairement à un interrupteur, il ne garde pas la position (il faut maintenir le doigt dessus pour qu'il fasse contact).



Le bouton poussoir est placé à cheval sur la ligne centrale de la plaque d'essai. D'où gain de place et placement dans le bon sens.

Ligne +

Montage :
La LED est connectée au pin 10. Sa sortie négative est branchée sur la plaque d'essai (ligne GND). Le bouton poussoir est autonome. Il est alimenté par la broche 5v. En sortie il est relié, à la fois, au pin2 **et** à une **résistance de 10 kΩ** laquelle retourne à la terre. Ici elle est dite montée en »pull-down* ».

Voir chapitre **Compléments, page 58, la définition du pull-down.*

Exo 1

Exercice n° 1 chapitre n° 4 : BlinkBP = reprise du sketch **BlinkBlink** du chapitre n° 3 avec rajout d'un bouton poussoir.

Matériel complémentaire : 1 bouton poussoir – 1 résistance de 10 k Ohms – fils de cablage.
Si le bouton poussoir est appuyé, le pin 10 sera mis sur HIGH par programmation, sinon il sera mis sur LOW.

Montage :
Il est présenté en page précédente et fait suite au montage du chapitre n° 3.
La LED reste connectée au pin 10 via la résistance qui peut être de 220 ou 330 Ohms.
Le bouton poussoir est monté. On branche le 5 v et le GND sur les lignes du bas de la breadBoard.

Une variable **pinBouton**, associée au bouton poussoir, est initialisée en mode INPUT.
On introduit la **fonction IF** pour tester l'état du bouton poussoir et n'allumer la LED que **si** le bouton est pressé. D'où une nouvelle variable : **etatBouton**

Attention au sens d'utilisation de la breadBoard. Par convention, la ligne du bas marquée **+** (généralement de couleur rouge) sera alimentée par le 5 volts et la ligne au-dessus (généralement de couleur bleue) sera branchée au GND.
La sortie de la LED est branchée sur la ligne de terre.

Le nouveau sketch prend le nom de **BlinkBP**.
Exécuter **tout de suite** la commande *Fichier/Enregistrer sous* avant de le téléverser.

Voici le nouveau programme :

```
/* BlinkBP  KonkArLab Nov 2017
breadBoard et LED et bouton-poussoir
*/
int led=10;  //variable pour le pin utilisé par la LED
int pinBouton = 2; //broche de connexion au bouton-poussoir
int etatBouton = 0; //variable pour stocker l'état du bouton

void setup()
{
    pinMode(led,OUTPUT); //la LED est en sortie de courant
    pinMode(pinBouton,INPUT); //le bouton-poussoir est une entrée de donnée
}

void loop()
{
//on lit l'état du bouton et on le stocke dans etatBouton
  etatBouton=digitalRead(pinBouton);
//
  if (etatBouton == HIGH) {     // ou if (etatBouton)
    digitalWrite(led,HIGH); // on passe le pin à +5V
   }
   else {
    digitalWrite(led,LOW); // on passe le pin à 0V
   }
}
```

Les ordres delay ne servent plus à rien et ont été supprimés.
On vérifie que la LED s'allume lorsqu'on presse le bouton poussoir et s'éteint dès que l'on cesse d'appuyer sur le bouton. Sinon, revoir les connexions !

La console ou moniteur-série en sortie

Permet à la carte Arduino **connectée à un PC** d'envoyer des messages à l'ordinateur pour les afficher en temps réel sur son propre écran.

Le codage :

```
//Le programme Console affiche un message sur le moniteur-série
long randNombre;
void setup() {
  Serial.begin(115200);  //initialisation de la communication à 115200 bauds
  //Envoi du message. Attention aux caractères accentués :
  Serial.println("Felicitations de Konk Ar Lab !");
}
void loop() {
  randNombre=random(10,150); //la fonction random extrait un nombre entre 10 et 150
  Serial.println(randNombre) ;  //affichage du nombre sur la console
  delay(1000);   //pour éviter un défilement trop rapide
}
```

Serial est une **bibliothèque** incluse dans l'IDE qui gère les commandes du moniteur-série.

L'ordre d'affichage peut s'écrire sous 2 formes :
println() : affichage avec passage à la ligne pour le message suivant.
print() : affichage des messages successifs en continu sur la même ligne.

L'affichage s'obtient en cliquant sur le bouton de commande du moniteur-série :

Remarque : les broches D0 et D1 de la carte Arduino sont exploitées par le moniteur. Elles doivent rester libres si on l'utilise dans le programme.

Exo 2

Exercice n° 2 chapitre n° 4 : On complète le sketch BlinkBP par l'affichage de l'état du bouton sur la console.

Rajout dans le setup() de la fonction : `Serial.begin(115200);`

Et ajout dans le loop() : `Serial.println(etatBouton);`

Le moniteur est réglé sur **115 200** bauds, vitesse recommandée : modification éventuelle de la vitesse du moniteur par la ligne baud à la base de l'écran ou par la commande *Outils/Moniteur.*

L'affichage fournit une suite de 1 ou une suite de 0 selon que l'on appuie ou pas sur le bouton poussoir.

Un affichage sous forme **graphique** peut s'obtenir par la commande ***Outils/Traceur série***. Ici, le traceur ne produirait que deux lignes droites horizontales.

Exercice n° 2 bis : Sketch **Console.**

Ce nouveau sketch consiste à recopier le code ci-dessous, présenté en page précédente.

On commence par une commande *Nouveau programme.*

La carte reste branchée sur l'ordinateur pour la communication avec le programme. On reste avec la breadBoard utilisée précédemment.

Codage :

```
//Le programme Console affiche un message sur le moniteur-série
long randNombre;
void setup() {
  Serial.begin(115200);  //initialisation de la communication à 115200 bauds
  //Envoi du message. Attention aux caractères accentués :
  Serial.println("Felicitations de Konk Ar Lab !");
}
void loop() {
  randNombre=random(10,150); //la fonction random extrait un nombre entre 10 et 150
  Serial.println(randNombre) ;  //affichage du nombre sur la console
  delay(1000);   //pour éviter un défilement trop rapide
}
```

La console ou moniteur-série en entrée

Permet au PC de transmettre à l'Arduino des informations qui peuvent être exploitées par le programme actif, par exemple des paramètres. Le message, tapé au clavier, est transmis en cliquant sur la commande Envoyer ou la touche Envoi.

Le codage :

```
/*Le programme ConsoleClavier affiche sur la console
 *la donnée tapée au clavier dans la fenêtre du moniteur
 * Création Konkarlab 2017
 */
void setup() {
 Serial.begin(115200); //paramètre de communication
 Serial.println("Entrez la donnée dans la fenêtre du moniteur.");
}
void loop() {
  string caracters = "";
  while (Serial.available()) {  //tant que des caractères sont en attente
  char c =Serial.read();  //la variable c recueille le caractère
  caracters = caracters + c;   //on concatène
  }
  Serial.print(caracters); //on écrit la série sur le moniteur
}
```

La fonction **Serial.available()** fournit le nombre de caractères transmis en attente d'être lus.
La fonction **Serial.read()** extrait le caractère suivant du message. Le type est « char ».

On ouvre la fenêtre du moniteur (loupe) en cliquant sur son bouton de commande :
La saisie des données se fait dans la ligne supérieure (64 caractères maximum).

Exo 3

Exercice n°3 chapitre n° 4 : - Sketch ConsoleClavier : on recopie le sketch présenté dans la page précédente et reproduit ci-dessous.
Matériel : carte Arduino.

```
/*Le programme ConsoleClavier affiche sur la console
 *la donnée tapée au clavier dans la fenêtre du moniteur
 * Création Konkarlab 2017
 */
void setup() {
 Serial.begin(115200); //paramètre de communication
 Serial.println("Entrez la donnée dans la fenêtre du moniteur.");
}

void loop() {
  string caracters = "";
  while (Serial.available()) {  //tant que des caractères sont en attente
  char c =Serial.read();  //la variable c recueille le caractère
  caracters = caracters + c;   //on concatène
  }
  Serial.print(caracters); //on écrit la série sur le moniteur
}
```

La fonction **Serial.available()** indique le nombre de caractères en attente d'être lus. Si des caractères sont en attente (le type est "char"), on utilise la fonction **Serial.read()** qui va lire le premier caractère en attente dans la chaîne. On peut stocker ce caractère, l'écrire, le traiter. Pour parcourir la chaîne entière, on utilise une boucle **while** jusqu'à atteindre la fin de la chaîne. A l'exécution, on ouvre la fenêtre du moniteur. On frappe un message au clavier dans la fenêtre du haut. Envoi par un clic sur la commande *Envoyer* du moniteur ou la touche **Envoi** du clavier. Le programme affiche sur le moniteur le message transmis à l'Arduino. On note que les caractères accentués ne sont pas traduits, l'Arduino utilisant la table des caractères ASCII.

Retour sommaire

INITIATION
A
ARDUINO

Travaux pratiques (partie 2) :

le potentiomètre – le servomoteur – le capteur de luminosité – le buzzer – le capteur de température – le capteur d'inclinaison

Le potentiomètre

Exemples :

Le potentiomètre est une résistance variable qui se règle manuellement avec un bouton ou une glissière. Il est ici rotatif. La patte centrale représente le curseur qui donne la valeur de la résistance.
Le potentiomètre est un capteur. Il se branche sur une **entrée analogique** de l'Arduino. La lecture se fait par l'instruction **analogRead.**
Pour une entrée analogique, aucune configuration de la broche n'est nécessaire.
De très nombreux capteurs sont basés sur le principe de résistance variable et se cablent presque de la même façon. Exemples : la cellule photo-électrique, le capteur de pression, etc.

Des exemples d'utilisation de potentiomètre sont fournis dans la bibliothèque Arduino :
Dans **01.Basics** avec le programme **AnalogReadSerial** (affiche la valeur du potentiomètre sur la console) et dans **03.Analog** avec **AnalogInput** et **AnalogInOutSerial** .
Le programme **AnalogInput** fait agir un potentiomètre sur le rythme des clignotements de la LED13 de test en modulant les délais d'attente.

Exo 1

CONSEIL : avant tout nouveau montage, il est utile de purger la carte Arduino (téléversement d'un sketch vide ou du sketch **Blink**, par exemple).

Exercice n° 1 chapitre n° 5 : procéder au montage du potentiomètre – entrée analogique
Rechercher le sketch situé dans Fichier/Exemples/**Analog/AnalogInput** et le télécharger.
Le potentiomètre fait varier la cadence des clignotements de la LED (ils sont + ou – rapprochés).

Matériel : plaque d'essai **-** potentiomètre – fils de connexion
Montage : le potentiomètre ou potard est une résistance.
Potard sur A0. LED facultative branchée sur borne 13 et GRD.

La page qui suit reproduit le sketch et fournit les éléments du montage.
On notera que les commandes delay(sensorValue) exploitent la valeur du capteur.

Rappel : Pour une entrée analogique, aucune configuration de la broche n'est nécessaire. La lecture du potard se fait par l'instruction **analogRead**.

On pourrait aussi brancher une LED (rouge par exemple) et sa résistance de 220 Ohms.
Note :
Dans les exemples Arduino, en 03.Analog, voir également le sketch **AnalogInOutSerial** avec LED sur la borne 9, sortie moniteur et commande delay(2). Noter la fonction **map** telle que :

```
outputValue = map(sensorValue, 0, 1023, 0,255);
```

Pour une donnée, **map** transforme une suite de valeurs en une autre suite de valeurs.

Ne pas démonter le potentiomètre qui servira pour le b) de l'exercice suivant.

Le programme AnalogInput :

```
/*
  Analog Input
 Demonstrates analog input by reading an analog sensor on analog pin
0 and turning on and off a light emitting diode(LED) connected to
digital pin 13.
 The amount of time the LED will be on and off depends on
 the value obtained by analogRead().

 The circuit:
 * Potentiometer attached to analog input 0
 * center pin of the potentiometer to the analog pin
 * one side pin (either one) to ground
 * the other side pin to +5V
 * LED anode (long leg) attached to digital output 13
 * LED cathode (short leg) attached to ground

*/

int sensorPin = A0;    // select the input pin for the potentiometer
int ledPin = 13;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from the
sensor

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}
void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```

Le servomoteur

Mini servomoteur SG90 Tower Pro :

Utilisé pour des mouvements de précision, le servomoteur permet de faire tourner le moteur d'un nombre de degrés précis. Il est largement utilisé en robotique pour animer des robots et en modélisme. En général, le servomoteur ne se déplace que sur un demi-tour. Le Tower Pro SG90, ci-contre, est réalisé avec des engrenages et des bras en plastique, ce qui le limite à des efforts de faible puissance.

Montage :
Le servomoteur comporte trois fils. Le rouge est branché sur le 5v de l'Arduino, le noir ou marron sur le GND et le blanc ou jaune (signal) sur une broche PWM (9 par exemple).

Code : La bibliothèque **Servo.h** fournit les commandes associées au servomoteur :

```
Servo nomVariable; // crée un objet servo

nomVariable.attach(9);  //associe l'objet à une broche, ici la 9

nomVariable.Write(pos);  //demande au servo d'aller à la position contenue dans la
//variable pos, variable entière dont les valeurs sont comprises entre 0 et 180.
```

Programmes (voir exo 2) : Le croquis **Exemples/Servo/Sweep** entraîne la rotation du moteur, par pas de 1, de 0 à 180 degrés, puis de 180 à 0 degré.
Le croquis **Exemples/Servo/Knob** utilise un potentiomètre pour commander le positionnement du moteur.

Exo 2

Exercice n° 2a chapitre n° 5 : Servomoteur
Matériel : plaque d'essai – servomoteur avec ses 3 fils
Montage : Les 3 fils du servomoteur sont branchés à l'Arduino: fil noir (ou blanc) à la terre, fil rouge au 5v et fil jaune à la borne n° 9. On emboîte un bras, **sans le visser**.

Programme : On charge le programme **Exemples/Servo/Sweep.** Il entraîne la rotation du moteur, par pas de 1, de 0 à 180 degrés, puis de 180 à 0 degré. On peut noter une utilisation intéressante de la commande **for**, représentée ci-dessous :

```
void loop() {
  for (pos = 0; pos <= 180; pos += 1) { //goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);     // tell servo to go to position in variable 'pos'
    delay(15);              // waits 15ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);     // tell servo to go to position in variable 'pos'
    delay(15);             // waits 15ms for the servo to reach the position
  }
}
```

Exercice n° 2b : Servomoteur + potentiomètre
Montage : néant si les 2 précédents montages sont conservés**.**
Programme : On charge le programme **Exemples/Servo/knob.** Le potard commande le positionnement du moteur (déplacement du bras).
L'instruction **map**(valeur, 0, 1023, 0, 179) transpose les valeurs du potard en valeurs pour le servomoteur.

Le capteur de luminosité

Une photorésistance permet de mesurer les variations de la luminosité (l'unité de mesure est le lux). Toutefois, la valeur fournie par ce type de capteur n'a pas d'unité, elle est purement indicative.
Elle peut être utilisée pour la détection jour/nuit ou celle d'une présence, la mesure de la luminosité ambiante (pour ajuster un éclairage par exemple), le suivi de lumière (pour panneaux solaires, robots, etc).

Quelques valeurs de luminosité :

Lieu concerné	Éclairement moyen
Nuit de pleine lune	0,5 lux
Rue de nuit bien éclairée	20 à 70 lux
Local de vie	100 à 200 lux
Appartement bien éclairé	200 à 400 lux
Local de travail	200 à 3 000 lux
Stade de nuit	150 à 1 500 lux
Extérieur par ciel couvert	500 à 25 000 lux
Extérieur en plein soleil	50 000 à 100 000 lux

Exemple d'application :

```
if (valeurCapteur < 10) {
    Serial.println(" - Noir");
  } else if (valeurCapteur < 200) {
    Serial.println(" - Sombre");
  } else if (valeurCapteur < 500) {
    Serial.println(" - Lumiere");
  } else if (valeurCapteur < 800) {
    Serial.println(" - Lumineux");
  } else {
    Serial.println(" - Tres lumineux");
  }
```

Exo 3

Exercice n° 3 chapitre n° 5 : Programme ConsoleLux : Capteur de luminosité

Matériel : plaque d'essai - capteur de lumière et résistance de 10k Ohms.

Montage (avec pont diviseur de tension*) : Le capteur (photorésistance) est monté sur la platine d'essai. Au voltmètre, il affiche 12,2 k Ohms. Les broches du capteur sont branchées sur des colonnes 1 et 2 de la plaque d'essai, une résistance de 10 K Ohms sur la colonne 2 et une colonne 3. La résistance additionnelle doit être au moins équivalente à la plus forte valeur de résistance du capteur, pour garantir une meilleure sensibilité de celui-ci.
La colonne 2 est connectée à la borne A0 de l'Arduino. Les colonnes 1 et 3 sont connectées respectivement à 5 V et à la masse, ou inversement.

Programme : Chargement du programme **Console**, créé au chapitre n° 4 : *Fichier/Carnet de croquis/Console*, puis *Fichier/Enregistrer sous* et on nomme le nouveau programme, par exemple, **ConsoleLux.**

Dans le programme, on remplace le contenu de la loop() par le suivant :

```
void loop() {
   int valeurCapteur = analogRead(A0);   //lecture du capteur
   Serial.println(valeurCapteur);  //affichage du nombre sur la console
   delay(100);          //pour éviter un défilement trop rapide
}
```

Téléversement et affichage sur le moniteur.

**voir la définition dans le volet COMPLEMENTS ; chapitre Résistances.*

Générateur de son ou bipeur

Générateur de son PCB #A125 :

Pour l'écoute, il faut retirer la pastille de papier.

Un **bipeur** (*buzzer* ou *beeper* en anglais) est un élément électromécanique ou piézoélectrique qui produit un son caractéristique quand on lui applique une tension (*continue ou alternative*) : le bip.
1) électromécanique, il nécessite une tension continue pour fonctionner. Il émet un son fixe.
2) piezoélectrique, il nécessite une tension alternative pour fonctionner. Il se compose d'un ***diaphragme*** piézoélectrique et d'une cavité avec un orifice. Le son varie en fonction d'une valeur de fréquence comprise entre 120 et 1500 hertz.
Il est encore actif à des tensions inférieures à 5V, ce qui le rend intéressant dans des montages d'alarme (bips) où les batteries sont déchargées.

Les applications du bipeur peuvent être, par exemple : bip du réveil, jouets , minuterie, signalisation sonore sur une machine, sur un appareil ménager, carillon de porte…etc.
Associé à un capteur de luminosité, il devient un instrument de musique !

Exo 4

Exercice n° 4 chapitre n° 5 : Complément au sketch **ConsoleLux** : ajout d'un buzzer qui émettra, en continu, des sons en fonction de la luminosité.
Il représente un instrument de musique appelé le Theremin, développé dans les années 1920 et qui fut l'un des premiers instruments électroniques : une main joue sur le volume et l'autre sur la hauteur du son.
Montage : HP sur la broche n°9 (l'aiguille la plus longue) et le GND (l'aiguille la plus courte).

Sketch utilisé : Croquis *Exemples/Digital/tonePitchFollower*.
Modifications :

```
Serial.Begin(115200);
tone(9, thisPitch, 10);  //le son est émis durant 10 ms pour être audible.
```

Annexe : quelques exemples d'utilisation du capteur de luminosité :

Commande de la lumière d'une pièce : éteindre automatiquement la lumière d'une pièce grâce à un relai contrôlant l'arrivé de courant de la lampe. Lorsque la lumière est forte on envoie un signal au relai pour qu'il se ferme ou reste fermé. Inversement, quand il fait sombre, il déclenche l'allumage de la lumière. Voir exemple *ici*.
(http://www.manuel-esteban.com/arduino-capteur-de-luminosite)
Pilotage de l'éclairage extérieur de la maison ou… d'une porte de poulailler. Voir *ici*
(https://electrotoile.eu/arduino-fabriquer-un-interrupteur-crepusculaire-avec-photoresistance-LDR.php)
Diverses applications de domotique. Voir *ici*.
(https://electrotoile.eu/domotique.php#jeedom)
Robot suiveur de lumière : un robot, par exemple, qui va se diriger vers la source la plus lumineuse. Voir *ici*.
(http://www.semageek.com/diy-comment-fabriquer-simplement-un-petit-robot-suiveur-de-lumiere)

Le capteur de température

Une thermistance permet de mesurer les variations de la température. Toutefois, la valeur fournie par ce type de capteur n'a pas d'unité, elle est purement indicative. La conversion en degrés Celsius s'obtient par des formules.

Montage :
Le montage est identique à celui du capteur de luminosité. On pourra donc utiliser le programme ConsoleLux, sans modification, pour tester le capteur.

Tableau des correspondances entre les valeurs indiquées par le capteur et les degrés Celsius (extraits) :

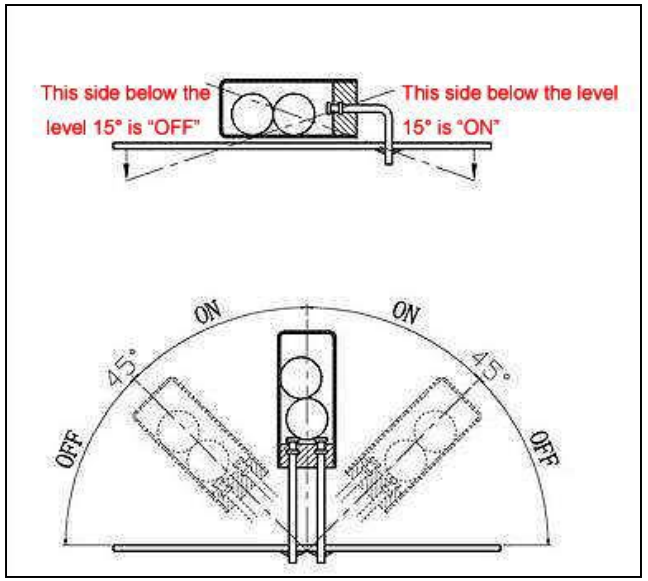
ADC	250	…	350	375	400	425	450	475	500	525	…	625	650	675	700
C	1.4	…	11.1	13.4	15.6	17.7	20.0	22.2	24.4	26.7	…	36.1	38.7	41.3	44.1

Le capteur d'inclinaison

Capteur SW-520D :

Le capteur d'inclinaison permet de détecter l'orientation ou l'inclinaison. Ce capteur se comporte comme un interrupteur. D'où l'appellation de «interrupteur à mercure», «interrupteur à bascule» ou «capteur à bille roulante», selon le modèle.
Sa simplicité le rend populaire pour les jouets, les gadgets et les appareils de jeu (tilt).

Le principe du fonctionnement du capteur SW-520D est donné par le schéma ci-contre :



Caractéristiques de fonctionnement : la position stationnaire, indiquée dans la partie ON, où le commutateur est dans un état de conduction, est rompue lorsque l'inclinaison tombe au dessous du niveau de 15 degrés. L'interrupteur est alors à l'état OFF et indique 0 volt.

Montage : il est identique à celui du bouton poussoir. Sur borne numérique et test sur le moniteur.

Retour sommaire

INITIATION
A
ARDUINO

Compléments :

Les fonctions – Rôle des résistances – Couleurs de fils – Bibliographie

Les fonctions (généralités)

Une fonction (également désignée sous le nom de procédure, de routine ou de sous-programme) est une suite d'ordres que l'on peut appeler à tout endroit du programme.
Le langage Arduino est constitué lui-même d'un nombre important de fonctions, par exemple : setup(), analogRead(), digitalWrite(), delay(), loop(), Serial.begin()…

La fonction est déclarée par son nom. Ce nom est précédé du mot « void » signifiant vide, quand la fonction ne fournit pas de réponse. Sinon, le mot est omis et la fonction se termine par `return`.

Il est possible de déclarer ses propres fonctions, par exemple dans le sketch BlinkBlink :

```
void clignoteLed(){
   digitalWrite (led, HIGH);  //led a été préalablement déclaré dans ce sketch
   delay (1000);
   digitalWrite (led, LOW);
   delay (1000);
}
```

Pour exécuter cette fonction, il suffit de taper la commande suivante dans le Loop() : `clignoteLed();`

On peut introduire des **paramètres** dans une fonction :

```
void clignoteLed(int led,int delai){   //déclaration de led et de delai
   pinMode(led,OUTPUT);  //La LED est configurée en sortie de courant
   digitalWrite (led, HIGH);   //on passe le pin à 5 volts
   delay (delai);   //pause
   digitalWrite (led, LOW);   //on passe le pin à 0 volts
   delay (delai);
}
```

Dans cet exemple, on modifie la durée du clignotement depuis la commande qui appelle la fonction :

```
clignoteLed(3,250); //la Led de la broche D3 clignotera lentement
```

***Remarque** : Les fonctions sont généralement écrites après la boucle du loop().*

1/3

Les fonctions (généralités)

Cette fonction clignoteLed() pourrait servir dans des sketches pour signaler une anomalie…
On va donc **l'enregistrer dans la librairie du répertoire Arduino** (l'emplacement du répertoire Arduino est spécifié dans *Fichier/Préférences*).

Pour ce faire, on va créer 2 fichiers, un fichier d'en-tête (avec extension h) et un fichier du code source (avec extension cpp). Le premier fichier contient la description de la fonction et de ses variables, le second fichier contient le code source du programme.

Les 2 fichiers sont au format texte et peuvent être créés à l'aide de Bloc-notes, l'éditeur de.Windows (ou l'éditeur de la distribution Linux).
Le fichier d'en-tête est nommé *clignoteLED.h* et il aura le contenu suivant :

```
/*fichier clignoteLed.h*/
#include "Arduino.h"
void clignoteLed(int led, int delai);
```

Le fichier du code source, nommé *clignoteLED.cpp* aura le contenu suivant :

```
/*fichier clignoteLed.cpp*/
La LED est connectée à la broche dont le numéro est fourni en paramètre
Le programme fait clignoter la LED

#include "Arduino.h"
#include "clignoteLed.h"
void clignoteLed(int led, int delai){
   pinMode(led,OUTPUT); //la LED est configurée en sortie de courant
   digitalWrite(led,HIGH); // on passe le pin à +5V
   delay(delai);    //pause
   digitalWrite(led,LOW); // on passe le pin à 0V
   delay(delai);
}
```

Les fonctions (généralités)

Pour le stockage de la fonction, on crée le dossier *clignoteLED* dans la librairie du répertoire *Arduino* et on y transfère les fichiers *clignoteLED.h* et *clignoteLED.cpp*.

Dans les programmes qui voudront allumer des LEDs, on introduira en début de sketch la commande `#include "clignoteLed.h`
et l'appel de la fonction dans le loop() avec ses paramètres sera de la forme :
`clignoteLed(10,1000);`

Le sketch suivant , contient une fonction `maFonctionX` qui retourne un résultat :

```
void setup() {
    Serial.begin(19200);
}
void loop() {
    int i=random(3,60);
    int j=3;
    float k;      //type décimal avec 7 chiffres après la virgule
    k = maFonctionX(i,j);   //appel de la fonction
    Serial.print(i);
    Serial.print("       Résultat =  ");
    Serial.println(k,2);  //affichage sur le moniteur avec 2 décimales
    delay(500);
}
//
float maFonctionX(int x, int y) { //float est le type de la donnée retournée
    float result;
    result = float(x) / y;            //float(x) pour calcul avec décimales
    return result;
}
```

La fonction retourne une valeur, d'où l'absence du mot « void ». Le type de la valeur de retour, ici de `float`, précède le nom de la fonction dans la déclaration de celle-ci.

Un aperçu du rôle des résistances

Les résistances électriques peuvent avoir plusieurs fonctions dans un circuit. Elles peuvent limiter un courant, ou encore amener un potentiel à un endroit en y faisant passer un minimum de courant…

1) La limitation de courant :

C'est l'utilisation la plus courante des résistances. Elle met en jeu la loi d'Ohm.
Un exemple a été présenté dans le chapitre n° 3 pour calculer la valeur d'alimentation d'une LED dont le voltage est donné d'environ 1,2 v et l'intensité de 20 mA :
On applique la loi d'Ohm selon la formule **R = U/I**
R = (5- 1,2)/0,020 = 190 Ohms
R représente la valeur minimum de la résistance à monter pour abaisser la tension d'alimentation.

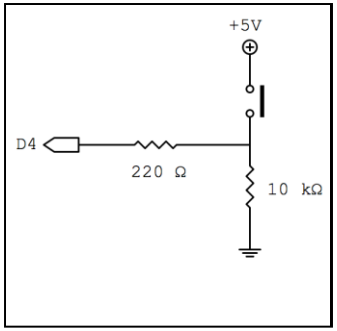
Un autre exemple sur l'alimentation d'un composant qui aurait besoin de 0.5A pour fonctionner se calcule de la façon suivante :
R = 5V / 0.5A = 10 Ohms
Une résistance de 10 Ohms permettra de fournir un courant de 0.5A au composant.

2) la variation de potentiel :

La résistance est utilisée pour permettre une lecture franche d'un conducteur.
Sans sa présence, le conducteur qui ne reçoit aucune tension est soumis à des parasites environnants qui le chargent à un potentiel inconnu (on parle de potentiel flottant) ce qui rend sa lecture aléatoire. Selon le montage, la résistance est dite de rappel ou de tirage.

a) La résistance de rappel (ou pull-down) : Une de ses entrées est fixée au potentiel bas, à savoir la masse (GND) ou le 0 V.
Le bouton poussoir présenté dans le chapitre n° 4 comporte ce type de montage.
La résistance permet de tirer le potentiel vers le bas (pull-down ou rappel au moins).

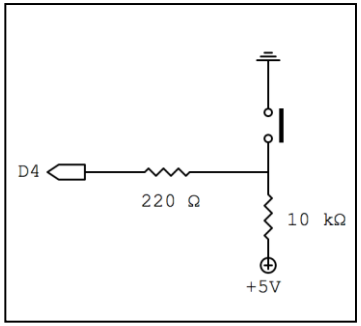
Le schéma ci-contre représente le montage du bouton poussoir :



La pression su le bouton fournit un courant électrique sur l'entrée. Le courant prendra le chemin le plus direct, donc par la résistance de 220 Ω et finira sur D4, qui est relié à la terre. Si on relâche le bouton, la résistance pull-down met l'entrée à la terre en l'absence de courant. Comme D4 est aussi relié à la même terre, il y a alors une différence de potentiel (une tension) de 0 Volts, et donc pas de signal parasite à l'entrée de D4.

b) La résistance de tirage (ou pull-up): une de ses entrées est fixée au potentiel haut, à savoir la source d'alimentation constituée par le 5V (ou 3,3 V) pour une carte Arduino.

Le schéma ci-contre inverse la position du bouton poussoir par rapport au schéma précédent. Le circuit étant ouvert, une tension de +5V est appliquée à l'entrée de la broche D4.



Une pression sur le bouton poussoir, fournit un courant électrique qui empruntera le chemin offrant le moins de résistance, donc directement par la masse (GND), sans passer par l'entrée de la broche D4.
Le fonctionnement est donc inversé par rapport à la résistance pull-down.

Comment choisir entre résistance pull-down ou pull-up ?

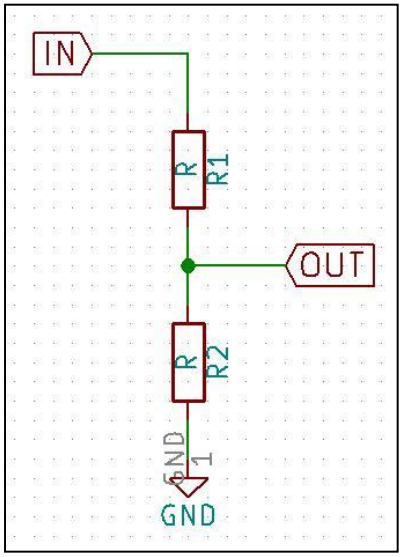
La différence entre une résistance pull-down et une résistance pull-up se traduit au niveau de la lecture de l'information :
Avec une résistance pull-down, l'entrée sur la broche est égale à 0, par défaut.
Avec une résistance pull-up, l'entrée sur la broche est égale à 1, par défaut.
Dans la condition : `if (digitalRead(bouton) == 1 )`, 1 sera la valeur d'entrée reconnue respectivement lorsque le circuit est fermé (pull-down) ou ouvert (pull-up).

3) Pont diviseur de tension

C'est un montage qui permet d'obtenir une tension proportionnelle à une autre tension.
Il est utilisé dans le chapitre n° 5 pour améliorer la sensibilité du capteur de luminosité.

La formule d'un pont diviseur de tension est la suivante :
Vout = Vin * (R2 / (R1 + R2))

Schéma d'un pont diviseur de tension :



Dans ce montage, la résistance R1 est remplacée par la photorésistance. Par conséquent, la valeur de R2 joue énormément sur la sensibilité du montage.

Si la valeur de R2 est négligeable par rapport à la valeur de R1, la photorésistance sera peu sensible, car il faudra atteindre un niveau de luminosité important avant que R2 s'approche de la valeur de R1 et commence à faire une différence dans l'équation.
Au contraire, si la valeur de R2 est non négligeable par rapport à la valeur de R1, la photorésistance sera très sensible car la moindre différence de luminosité entrainera un changement de la tension en sortie du pont diviseur.

Pour résumer, plus la valeur de la résistance R2 est grande, plus la photorésistance est sensible. A l'inverse, plus la valeur de la résistance R2 est faible, moins la photorésistance est sensible. Trouver le juste milieu peut prendre un certain temps. Dans ce genre de cas, un potentiomètre en lieu et place de R2 s’avère très pratique.

4) Le calcul de résistances équivalentes

Dans un circuit comprenant plusieurs résistances, il peut être important de connaitre la valeur R de la résistance équivalente.
La formule de calcul est différente selon que le circuit est en série ou en parallèle :

a) Résistances en série : les résistances sont placées les unes à la suite des autres dans le circuit.
Formule :
R = R1 + R2 + R3
Soit 3 résistances en série : R1, R2 et R3, respectivement de 100, 220 et 330 Ohms.
La résistance équivalente R aura pour valeur :
R = 100 + 220 + 330 = 650 Ohms

b) Résistances en parallèle : Les bornes de même nature des résistances sont reliées ensemble (les "+" ensemble et les "–" ensemble).
Formule :
R = 1/R1 + 1/R2 + 1/R3 = 1 / (R1 + R2 + R3)

Compléments :
- Il existe des applications sur smartphone pour connaître la valeur de la résistance.
Exemple : **Resistor Color Code**.
- Les fiches d'information (datasheets) des composants fournissent leurs caractéristiques.
Elles sont accessibles sur divers sites. Par exemple : www.datasheet.fr

A propos de la couleur des fils

Lors des câblages et en fonction des choix de couleurs de fils dont on dispose, il peut être utile de se fixer des règles dans le choix des couleurs :
Un exemple est donné ci-dessous :

Type de connexion	Couleur de choix
au **+** alimentation 3,3 ou 5 volts	rouge
au **-** masse (GND)	noir - bleue
à du 0 volt	noir
à sortie de la carte Arduino (émission/commande)	jaune
à entrée dans la carte Arduino (réception/lecture état)	vert
divers	blanc, gris, violet…

A noter que le marron n'existe pas dans Fritzing pour le dessin des cablages.

Bibliographie

Ce tutoriel s'est inspiré de plusieurs sites de référence parmi lesquels :
https://Arduino.education : « Arduino à l'école, apprendre en bidouillant ». Cours prévu pour des élèves de 13 à 16 ans. Fournit un apprentissage des bases de l'électronique et du code informatique. Mis à jour en 2020.
http://pierrecaulet.free.fr/Documentation/__%20arduino.pdf : manuel de 115 pages sur Arduino. De 2011.
https://zestedesavoir.com/ : Tuto « Arduino : premiers pas en informatique embarquée ». Maj 2020.

Autres sites :
https://www.arduino-france.com/ : tutoriels, symboles électroniques des composants. Maj 2023.
http://locoduino.org/ : Applications nombreuses. Orienté vers le modélisme ferroviaire. Maj 2023.
https://www.cours-gratuit.com/cours-arduino/ : Tout en matière de tutos.
https://**arduino.blaisepascal.fr/** L'essentiel.
https://**electroniqueamateur.blogspot.com/** de l'Arduino à l'ESP
http://**tiptopboards.free.fr/arduino_forum/index.php** des tutoriels pour Arduino.
https://plaisirarduino.fr/ Plusieurs tutos sur Arduino.
http://**fablab37110.ovh/doku.php?id=start:arduino:esp32:lorawan** : Sur Arduino, ESP32, Impr 3D, drones, etc.
https://**arduinogetstarted.com/arduino-tutorials** : tutoriels.

Sites déjà cités dans le tutoriel :
https://www.arduino.cc : le site Arduino (en anglais).
www.datasheet.fr : site sur les datasheets de composants.

Certains de ces sites peuvent disparaître avec le temps. Indulgence requise…

Liste actualisée le 08/04/2023.

Retour