Arduino - Initiation

Introduction aux microcontrôleurs

Arduino est la marque d'une plateforme de prototypage open-source qui permet aux utilisateurs de créer des objets électroniques interactifs à partir de cartes électroniques équipées d'un ***microcontrôleur***.

Un ***microcontrôleur*** est un ***circuit intégré*** qui intègre les éléments essentiels d'un ordinateur : processeur, unités périphériques et interfaces d'entrées-sorties. Les microcontrôleurs se caractérisent par un plus haut degré d'intégration (taille réduite), une plus faible consommation électrique et un coût réduit par rapport aux microprocesseurs polyvalents utilisés dans les ordinateurs personnels.

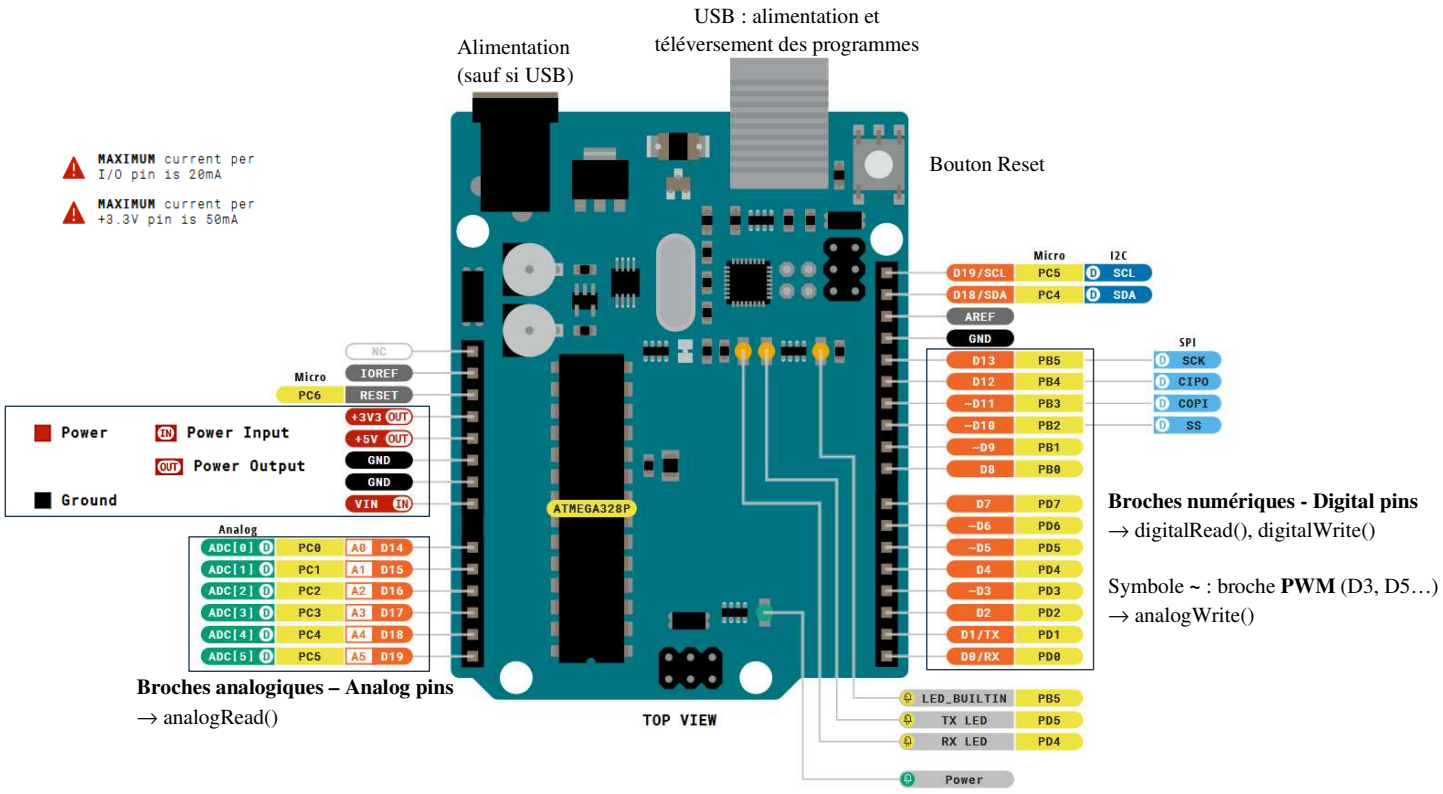
Un microcontrôleur peut être programmé pour analyser et produire des signaux électriques.
Ils sont utilisés dans les ***systèmes embarqués*** pour piloter des robots, dans les voitures, les avions, les récepteurs GPS, les télécommandes, l'électroménager, les jouets, la téléphonie mobile, la domotique, etc.

Dans le cas d'Arduino, les langages de programmation utilisés sont **C** et **C++**.
Un **IDE** (environnement de Développement Intégré) Arduino permet d'écrire et de téléverser les programmes dans la mémoire ROM du microcontrôleur (cf. ci-dessous).
Rq : Pyzo ou Spyder sont des IDE pour Python, VisualStudio est un IDE multi langages.

Le microcontrôleur des cartes Arduino utilise deux types de mémoires.

- ✓ Une ***mémoire morte*** (**ROM**, pour l'anglais read-only memory), une mémoire au contenu non volatile utilisée pour enregistrer des informations qui doivent être conservées lorsque l'appareil qui les utilise n'est plus sous tension.
  La carte Arduino utilise une **EEPROM** (Electrically-Erasable Programmable Read-Only Memory ou mémoire morte effaçable électriquement et programmable) qui peut être facilement effacée et réécrite à l'aide d'un courant électrique.
  C'est dans cette mémoire que seront stockés les programmes.
- ✓ Une ***mémoire vive*** (**RAM** acronyme anglais pour random-access memory), « mémoire à accès aléatoire » à accès rapide dans laquelle peuvent être enregistrées des données volatiles.

La carte Arduino Uno - Brochage



La modulation **PWM** (Pulse Width Modulation en anglais) ou modulation de largeur d'impulsions (MLI en français) est une technique utilisée pour synthétiser des signaux pseudo analogiques à l'aide de circuits numériques.

Instructions pour lire le signal appliqué à une broche :

- ✓ **`analogRead()`** sur une entrée ***analogique*** (A0 à A5) (tension délivrée par un capteur par exemple) ;
- ✓ **`digitalRead()`** sur une entrée ***numérique*** (D2, D4, D7, D8, D12, D13).

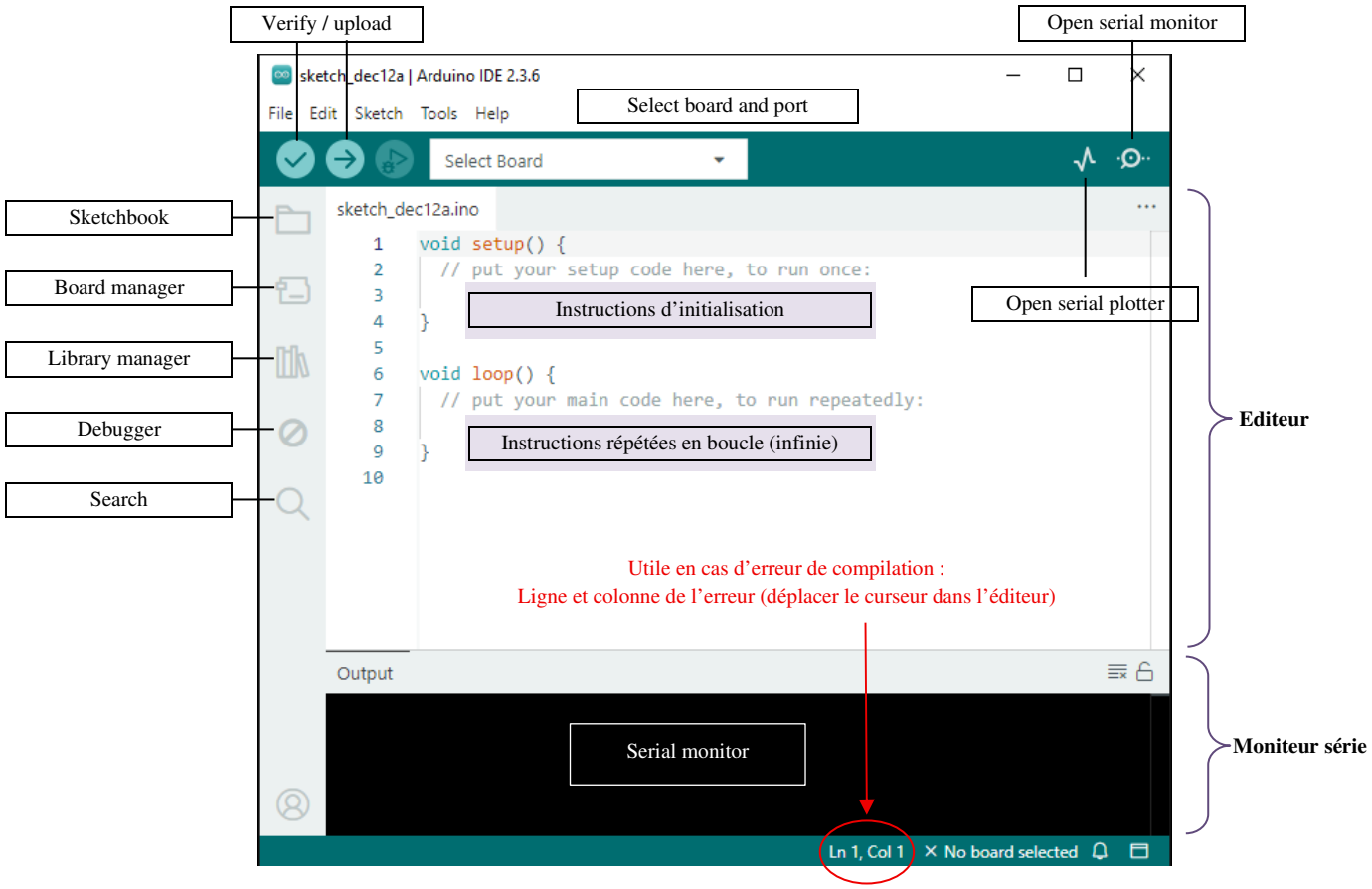
Instructions pour envoyer un signal à une broche :

- ✓ **`analogWrite()`** sur une broche ***numérique*** **PWM** simulant un signal ***analogique*** (D3, D5, D6, D9, D10, D11) ;
- ✓ **`digitalWrite()`** sur une broche ***numérique***.

Définir le comportement d'une broche (entrée ou sortie) : **`pinMode(`***`broche, mode`***`)`** avec *`mode`* = `INPUT` ou `OUTPUT`.

Sélectionner le type de carte Arduino et le port sur lequel elle est connectée dans la liste déroulante « Select Board ».

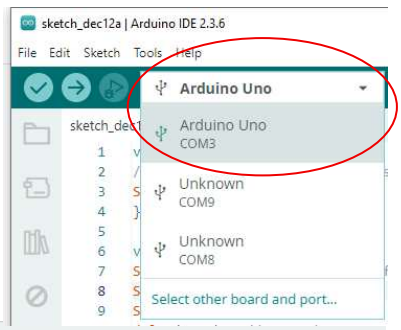
Un programme Arduino est appelé « ***sketch*** » (parfois traduit par « croquis » ou « esquisse »), l'extension de fichier est **.ino**
Il est tapé dans la zone d'édition de l'interface, il doit être ***compilé*** (transformation du code source en fichier binaire) puis téléversé sur la carte.
Des informations sur le processus de compilation sont affichées dans l'onglet « Output » sous l'éditeur (éventuelles erreurs, statut final, mémoire disponible…).

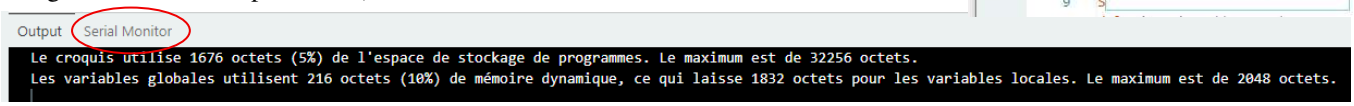


Sketchbook : programmes enregistrés sur l'ordinateur.
Board manager : packages nécessaires avec certaines cartes d'extension ajoutées sur la carte Arduino (WiFi,…).
Library manager : librairies/modules à utiliser pour les capteurs par exemple.
Serial moniter : visualisation du flux de données issu de la carte.
Les instructions **Serial.print()** et **Serial.println()** permettent d'écrire dans le moniteur série.
Serial Plotter : visualisation de graphes.

Utilisation d'une carte avec un ordinateur

1. Brancher la carte et lancer l'IDE.
2. Sélectionner la carte et le port dans la liste déroulante (cf. ci-contre).
   En cas de non détection, essayer de changer de port et de relancer l'IDE.
3. Créer un nouveau sketch (File / New Sketch) ou ouvrir un sketch enregistré (le dernier utilisé est préchargé).
4. Taper ou modifier le code ; l'aide en ligne est abondante.
5. Compiler le code .
   Si tout se passe bien un message s'affiche en blanc (sinon lire le message affiché en rouge et remédier aux problèmes) :





6. Téléverser le code .
7. Cliquer sur l'icône du moniteur série ou sur l'onglet « Serial Monitor » s'il est déjà ouvert.

```
1  // Commentaire sur une ligne
2
3  void setup() {
4  // Initialisation : instructions exécutées une seule fois
5  Instruction1;
6  Instruction2;
7  }
8
9  void loop() {
10 // Instructions répétées en boucle infinie
11 Instruction3;
12 }
```

void : en C++, le mot-clé **void** indique qu'une fonction ne renvoie pas de valeur (cf. autres exemples de fonctions page 11).

Les **délimiteurs** sont les **accolades** (analogue à l'indentation en python).
⚠ Erreur : Compilation error: expected '}' at end of input (par exemple).

Chaque instruction doit se terminer par un point-virgule (en python, uniquement pour séparer 2 instructions sur une même ligne).
⚠ Erreur : Compilation error: expected ';' before ….

Commentaires sur une seule ligne : **//** (# en python) ; multilignes **/* … */** (""" … """ en python).

Noms de variables

Caractères utilisables : _, 0, 1, 2, ..., 9, A, B, ..., Z, a, b, ..., z (commence toujours par une lettre ou un tiret bas « _ »).

Types

Entiers :	**short**	(2 octets $-2^{-15} \le n \le 2^{15}-1$, 16 bits)	$2^{15} = 32\,768$
	unsigned short	(2 octets $0 \le n \le 2^{16}-1$)	
	char	(1 octet $-2^{-7} \le n \le 2^{7}-1$, 8 bits) ('A' et 65 sont équivalents, codage ASCII)	
	byte	(1 octet $0 \le n \le 2^{8}-1$)	
	int, unsigned int	dépend du microcontrôleur (16 bits sur Arduino Uno)	
	long	(4 octets $-2^{-31} \le n \le 2^{32}-1$, 32 bits)	
	unsigned long	(4 octets $0 \le n \le 2^{32}-1$)	
	bool ou **boolean**	deux valeurs **true** ou **false** (sans majuscule au contraire de python)	
Flottants :	float	($9{,}4039548.10^{-38}$ à $3{,}4028235.10^{+38}$, 32 bits)	Ex : 1.5e-2
Chaînes :	https://docs.arduino.cc/language-reference/en/variables/data-types/string/		

Affectation

Comme en python, le symbole d'affectation est le signe égal : « **=** ».

Déclaration des variables et types

Dans le langage C le type et les variables doivent être déclarés.
⚠ Erreur : '*nom_variable*' was not declared in this scope.

```
int i;                  // Déclaration sans affectation (l'affectation aura lieu plus loin dans le programme)
float pi = 3.14;        // Déclaration et affectation simultanées
char c = 'a';
```

Opérateurs

+, −, *, /, % (reste division entière)
== (test d'égalité), != (test de différence), <, >, <=, >= (tests de comparaison)
! (négation), || (ou), && (et)

Raccourcis

```
i=i+1;    ⇔    i++;        i=i-1;    ⇔    i--;
a=a+b;    ⇔    a+=b;       a=a-b;    ⇔    a-=b;
```

Instruction conditionnelle

```
if (conditions) {instructions}
else if (conditions) {instructions}          (instruction else if facultative, peut être répétée)
else (conditions) {instructions}             (instruction else facultative)
```

Rq : il existe une instruction conditionnelle multiple **switch…case**

Boucles

```
while (conditions) {instructions}
for (i=0;i<100;i=i+1) {instructions}         (par exemple)
```

Rq : il existe une boucle **do…while**

Tableaux

La taille des tableaux doit être une constante.

```
type nom_tableau[dimension]
```

Exemples :

```
int liste[10];
float L[3] = {1.1, 2.2, 3.3};
```

Accès via l'indice/index (la numérotation commence à 0 comme en python) : `nom_tableau[indice]`

Directives de compilation

Une directive de compilation indique au compilateur de procéder à des opérations préalables au début de la compilation.
Ces directives se situent en tout début du programme source.

```
#include <fichier>      // Inclure des librairies (analogue à import en python)
#include "fichier.h"    // Autre syntaxe
#define alias valeur    // Remplace alias par valeur
```

Exemple : `#define capteur1 A0` analogue à `const int capteur1 = A0` mais gain de mémoire avec define.

Const

Ce mot clé permet de définir une variable qui, une fois initialisée, ne pourra plus être modifiée.
Exemple :

```
Const float pi = 3.1415 ;
x = 2 * pi ;
```

Aide en ligne	https://docs.arduino.cc/language-reference/
Fonctions	https://docs.arduino.cc/language-reference/#functions
Variables	https://docs.arduino.cc/language-reference/#variables
Structure	https://docs.arduino.cc/language-reference/#structure

Transfert des données - Annexes techniques

Port série – Cf. Exemple 1

Un ***port série***, également appelé port COM, est un type d'interface informatique qui permet la communication entre un ordinateur et des périphériques externes. Il s'agit d'un port physique sur un ordinateur ou un appareil qui permet d'envoyer et de recevoir des données bit par bit et séquentiellement sur un seul fil.

Ouverture et écriture dans le port série

`Serial.begin(v)`	Ouvrir le port série et fixer la vitesse *v* de transmission (valeurs prédéfinies, cf. exemple 1).
`Serial.print(s)`	Ecrire la chaîne *s* sur la ligne courante (à la suite de la dernière chaîne écrite si elle existe) ou sur une nouvelle ligne sinon sans retour à la ligne.
`Serial.println(s)`	Ecrire la chaîne *s* sur la ligne courante (à la suite de la dernière chaîne écrite si elle existe) ou sur une nouvelle ligne sinon puis retour à la ligne.

Time

Il est parfois nécessaire d'indiquer au microcontrôleur un délai d'attente (entre une mesure et son traitement par exemple).

`delay(n)`	Attendre *n* ms (millisecondes).
`millis()`	Renvoie le nombre de millisecondes écoulées depuis que le sketch a été téléversé et exécuté.

Fichiers et caractères de codage « invisibles » – Cf. Exemple 2

Les fichiers comportent des caractères « invisibles » (on peut les visualiser dans un éditeur de texte tel que Notepad++).
Il s'agit par exemple des caractères qui provoquent un retour à la ligne, en python : \n (système Linux), \r\n (Windows), \r (Mac).
La bibliothèque python ***serial*** (https://pyserial.readthedocs.io/) permet de lire les lignes écrites dans le port série :
ligne = serial.Serial(*port_serie*, *vitesse*). readline() (lecture et stockage du résultat dans une variable nommée *ligne*).
Les données lues dans le port série sont au ***format binaire***, *ligne*.decode("utf-8") permet de décoder ce format.

Ces informations sont utiles lorsqu'il s'agit de sauvegarder des données lues sur le port série dans un fichier texte.

Les exemples ci-dessous doivent être considérés comme des programmes élémentaires destinés à être assemblés afin de réaliser la tâche souhaitée (on trouve de très nombreux exemples en ligne et/ou dans les exemples fournis avec l'IDE Arduino).
Chaque exemple illustre une fonctionnalité : communication, récupération des données, acquisition des données…
Toujours procéder par étapes en testant le bon fonctionnement des programmes au fur et à mesure de l'avancement.

Exemple 1 - Ecrire dans le port série

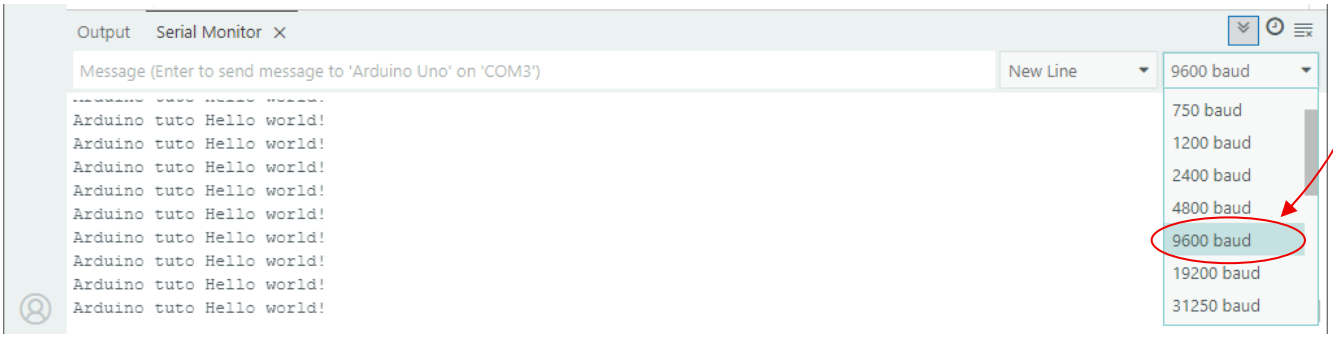
Application : visualiser des données en cours d'acquisition (acquisitions réelles à suivre)

Le skectch suivant illustre la communication entre la carte Arduino et l'ordinateur via le ***port série*** (physiquement, via le câble USB).

`Serial.begin(v)` Ouvrir le port série et fixer la vitesse *v* de transmission (valeurs prédéfinies, cf. ci-dessous).
`Serial.print(s)` Ecrire la chaîne *s* (ligne courante si existante ou nouvelle ligne sinon) sans retour à la ligne final.
`Serial.println(s)` Ecrire la chaîne *s* (ligne courante si existante ou nouvelle ligne sinon) puis retour à la ligne.
`delay(n)` Attendre *n* ms (millisecondes).

```
1  /*
2                    Exemple 1 - Ecrire dans le port série
3  */
4
5  void setup() {
6  // Ouverture port série - vitesse transmission (en baud = bit/s)
7  Serial.begin(9600);
8  // Tant que le port série n'est pas prêt, attendre 100 ms
9  while (!Serial) {
10       delay(100);     // Attendre 100 ms
11     }
12 }
13
14 void loop() {
15 Serial.print("Arduino "); // Affichages les uns …
16 Serial.print("tuto "); // … à la suite des autres (même ligne)
17 Serial.println("Hello world!"); // Affichage puis retour à la ligne
18 delay(1000); // Attendre 1000 ms
19 }
```

⚠ **Vérifier que la vitesse de transmission indiquée dans le port série correspond à la vitesse indiquée dans le sketch sous peine de ne rien voir s'afficher**.



Exemple 2 - Ecrire des valeurs numériques dans le port série et les stocker dans un fichier via python

Application : visualiser des données en cours d'acquisition et les récupérer via python (simulation, acquisitions réelles à suivre)

Le skectch suivant écrit des lignes (destinées à être lues par un script python annexe et enregistrées dans un fichier) dans le port série :
- une ligne d'en-tête (intitulés des colonnes) ;
- des lignes de données simulées (instant de mesure, valeur calculée à cet instant et numéro de la mesure).

Les lignes sont formatées au format csv : chaque colonne de données est séparée de la suivante par un ";".
Ces informations sont envoyées au port série et lues par un script python.

`millis()` Renvoie le nombre de millisecondes écoulées depuis que le sketch a été téléversé et exécuté.

```ino
int t0;                 // Instant initial, début des mesures (en ms)
int ti;                 // Instant mesure courante (n°i) (en ms)
float t;                // Durée calculée en s : t = (ti-t0)/1000.0
int cpt;                // Compteur (n° mesure courante)
int tMax = 5000;        // Durée de l'acquisition en ms
float mesure_capteur;   // Valeur calculée simulant une mesure via un capteur

void setup() {
    Serial.begin(115200);
    while (!Serial) {
      delay(100);
    }
    // En-tête futur fichier CSV (séparateurs colonnes ";") (script python annexe)
    Serial.println("ti (s);Capteur (V);Numero mesure;");
}
void loop() {
    cpt = 0;
    t0 = millis();
    ti = t0;
    while ((ti-t0) <= tMax) {           // Boucle de « mesure »
        delay(100);                     // délai de 100 ms entre les mesures
        cpt += 1;                       // N° mesure
        ti = (millis()-t0);             // Temps écoulé (en ms)
        t = (ti-t0)/1000.0;             // Temps écoulé (en s)
        mesure_capteur = 20 * t + 100;

        // Formatage au format CSV dans le moniteur série (« ; » entre colonnes)
        Serial.print(t);                // 1ère colonne de données
        Serial.print(";");              // Séparateur de colonnes
        Serial.print(mesure_capteur);   // 2ème colonne de données
        Serial.print(";");              // Séparateur de colonnes
        Serial.println(cpt);            // 3ème colonne de données
    }
    // Fin de la boucle while simulant les mesures
    Serial.println("Stop"); // A ce signal, le script python ferme le fichier.
    // Boucle infinie sans action : plus rien n'est envoyé sur le port série.
    while (true) {;}
}
```

Ligne 37 : la boucle infinie n'exécutant aucune action permet d'interrompre le flux de données transmises au port série (sinon l'envoi de la chaîne « Stop » est répété indéfiniment).

Programme Python

```py
import serial
import time

# Communication
port_serie = 'COM3'             # Cf. Arduino IDE (port sélectionné)
bauds = 115200                  # Cf. sketch arduino : Serial.begin(bauds)
# Enregistrement des mesures
dossier = ""                    # Dossier courant (dossier de ce fichier python)
nom_fichier = "exemple2.txt"    # A personnaliser
chemin = dossier + nom_fichier

ps = serial.Serial(port_serie, bauds) # Ouverture du port série
fichier = open(chemin, "w+")          # Ouverture du fichier en écriture
while True:                           # Boucle "infinie"
    ligne = ps.readline()             # Lecture d'une ligne sur le port série
    #print(ligne)                     # Débogage (fin de ligne = \r\n => 2 sauts)
    ligne = ligne[:-1]                # Traitement (suppression \n fin de ligne)
    ligne = ligne.decode("utf-8")     # readline -> binaire, conversion
    if 'Stop' in ligne:               # Message d'arrêt défini dans sketch Arduino
        break                         # Sortie de la boucle while
    print(ligne)                      # Vérification visuelle dans le shell
    fichier.write(ligne)              # Ecriture de la ligne dans le fichier
fichier.close()                       # Fermeture du fichier
ps.close()                            # Fermeture du port série
```

1. Téléverser le sketch sur la carte Arduino.
2. Fermer le moniteur série Arduino (ne pas déconnecter la carte du port USB).
3. Exécuter le programme python : les données lues doivent s'afficher dans le shell.

⚠ **L'onglet "Serial Monitor" doit rester fermé pour que python puisse ouvrir le port série**.
Erreur python : serial.serialutil.SerialException: could not open port 'COM3': PermissionError(13, 'Accès refusé.', None, 5)

https://docs.arduino.cc/built-in-examples/basics/ReadAnalogVoltage/

Application : enregistrer un flux ininterrompu de données analogiques

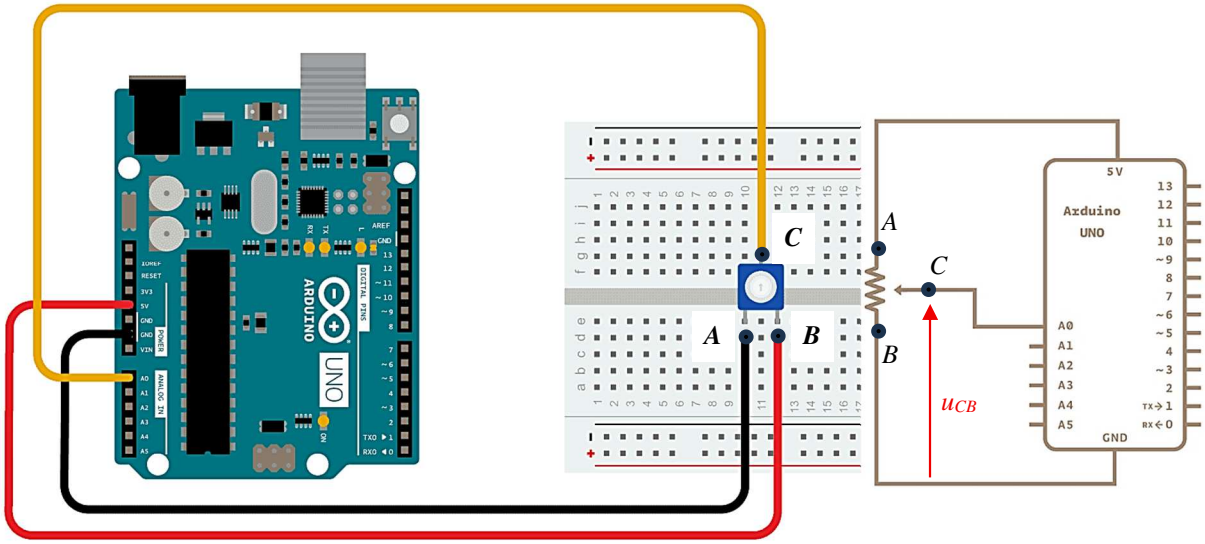
On applique sur la broche A0 de la carte Arduino la tension u_{CB} entre les pattes B et C d'un potentiomètre soumis à la tension u_{AB} = 5 V. On fait varier cette tension manuellement en agissant sur le potentiomètre rotatif.

`analogRead(pin)` Lire la valeur binaire sur la broche *pin* (A0 à A5), cf. page 1.

Convertisseur analogique numérique (CAN ou ADC pour analog-to-digital-converter en anglais) *n* bits : une grandeur analogique appliquée au CAN est discrétisée sur 2^n valeurs (de 0 à 2^n -1).

La carte Arduino UNO est équipée d'un CAN 10 bits : on dispose donc de 2^{10} = 1024 valeurs, de 0 à 1023, pour représenter la grandeur analogique appliquée sur une broche. ***La fonction `analogRead()` renvoie la valeur binaire issue du CAN***.
Il faut donc ***convertir*** la valeur binaire en volts dans le cas envisagé en sachant que la tension appliquée au potentiomètre est 5 V.

Schéma du circuit
La broche utilisée est la broche A0, on applique une tension au potentiomètre (10 kΩ) grâce aux broches 5V et GND de la carte.



Sketch

```
1  int U_binaire; // "Tension" lue : valeur renvoyée par analogRead() entre 0 et 1023
2  float U_Volts; // Tension vraie en volts
3
4  void setup() {
5      Serial.begin(115200);
6      while (!Serial) {
7        delay(100);
8      }
9  }
10
11 void loop() {
12   // Lecture de la tension analogique appliquée sur la broche A0 de l'Arduino
13   U_binaire = analogRead(A0);
14
15   // Conversion en Volts (analogique de 0 à 5 V ⇔ numérique de 0 à 1023)
16   U_Volts = U_binaire / 1023.0 * 5.0;
17
18   Serial.println(U_Volts);
19
20   delay(500);  // Utile pour ralentir le flux de données
21 }
```

La conversion valeur binaire / volts est effectuée ligne 16 (proportionnalité).
⚠ Ligne 16 : bien écrire **1023.0** et **5.0** afin que ces nombres soient traités comme des flottants

Remarque : le flux est ininterrompu, on verra dans d'autres exemples comment déclencher et interrompre le flux de mesures.

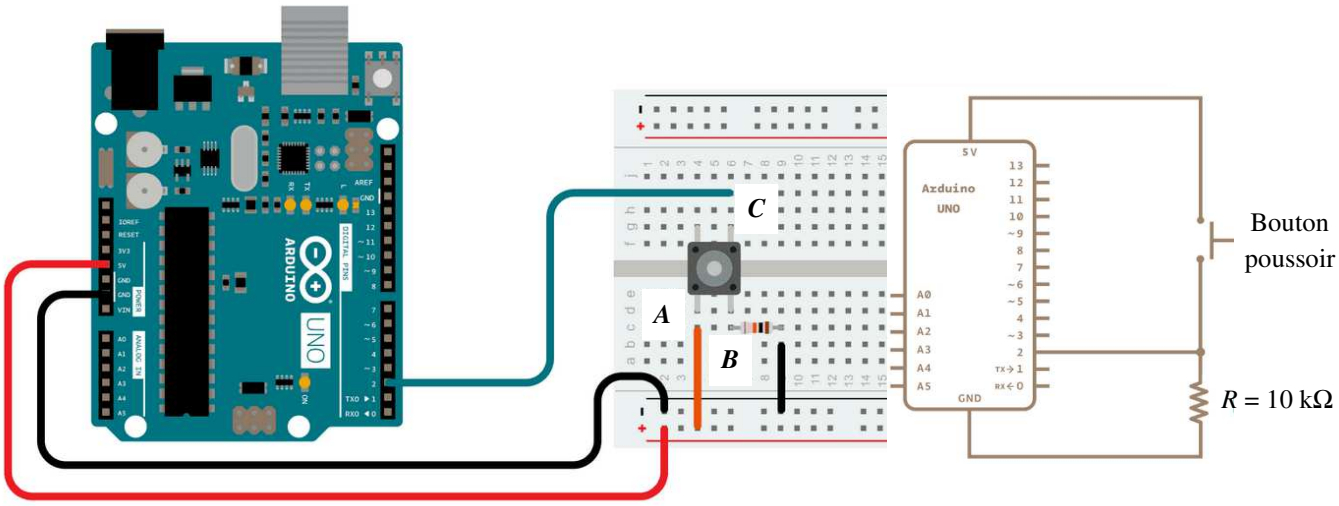
https://docs.arduino.cc/built-in-examples/basics/DigitalReadSerial/

Application : enregistrer un flux ininterrompu de données numériques

Bouton poussoir :
- lorsque le bouton est relâché, la patte C du bouton est reliée à B donc à la terre via la résistance *R*, la valeur LOW = 0 est alors lue par une broche numérique ;
- lorsque le bouton est enfoncé, la patte C du bouton est reliée au potentiel 5V en A, la valeur lue par la broche numérique correspond alors à la valeur HIGH = 1.

`pinMode(pin, mode)`	Configurer la broche *pin* en entrée (*mode* = INPUT) ou en sortie (*mode* = OUTPUT).
`digitalRead(pin)`	Lire la valeur booléenne sur la broche *pin* (2, 4, 7, 8, 12, 13).

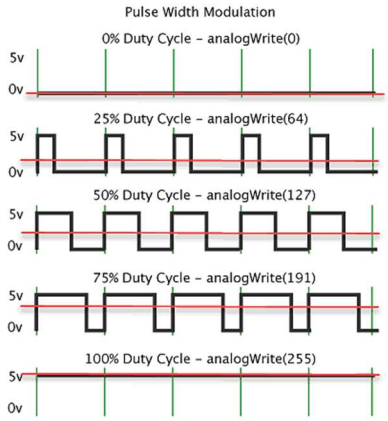
Schéma du circuit
La broche utilisée est la broche 2.



Sketch

```
1  int boutonPoussoir = 2;  // Nom de variable : bouton poussoir connecté broche 2
2  int etatBouton;
3
4  void setup() {
5      Serial.begin(115200);
6      while (!Serial) {
7        delay(100);
8      }
9      // Configure la broche 2 = boutonPoussoir en entrée (INPUT)
10     pinMode(boutonPoussoir, INPUT);
11 }
12
13 void loop() {
14   // Lecture de la broche numérique
15   etatBouton = digitalRead(boutonPoussoir);
16
17   Serial.println(etatBouton);
18   delay(500);
19 }
```

https://docs.arduino.cc/built-in-examples/basics/Fade/



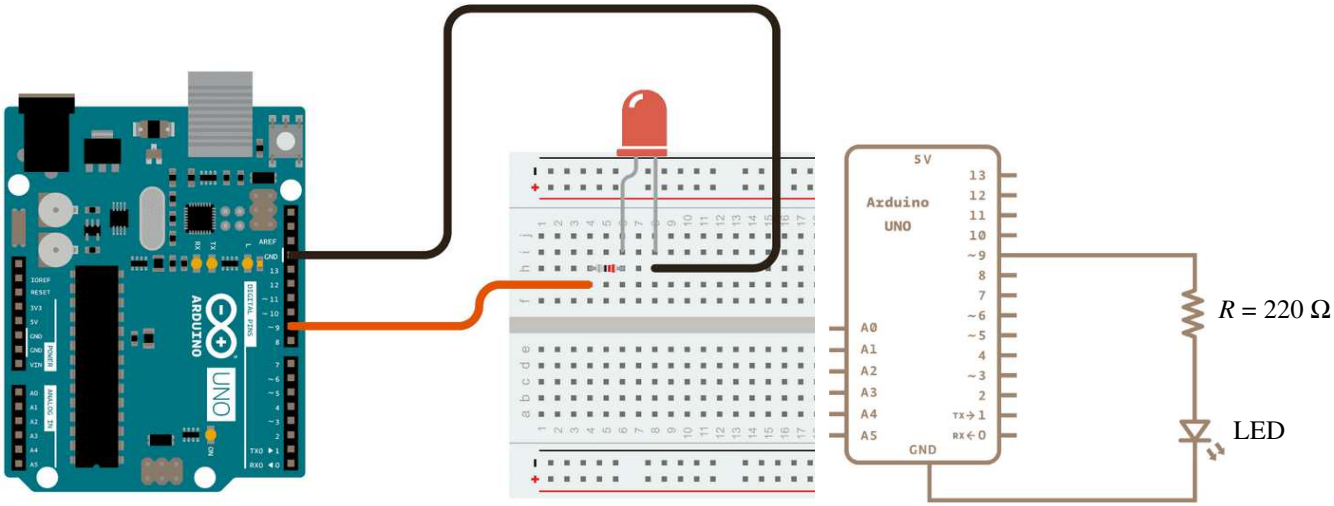
Application : générer un signal pseudo-analogique

Modulation de largeur d'impulsion (schéma ci-contre) : en faisant varier le ***rapport cyclique*** (durée du niveau haut / durée du niveau bas) des impulsions, on fait varier la ***valeur moyenne*** du signal.
C'est cette valeur moyenne qui simule un signal analogique. Le rapport cyclique pouvant varier rapidement, la valeur moyenne peut également évoluer au cours du temps et simuler un signal analogique variable dans le temps.

On fait varier la luminosité d'une LED en utilisant une sortie numérique PWM à modulation de largeur d'impulsion.

`analogWrite(pin, niveau)` Ecrire la valeur *niveau* sur la broche *pin* (~3, ~5, ~6, ~9, ~10, ~11 PWM = symbole ~).
`analogWrite()` accepte des valeurs entre 0 et 255.

Schéma du circuit
La broche PWM utilisée est la broche ~9.



Sketch

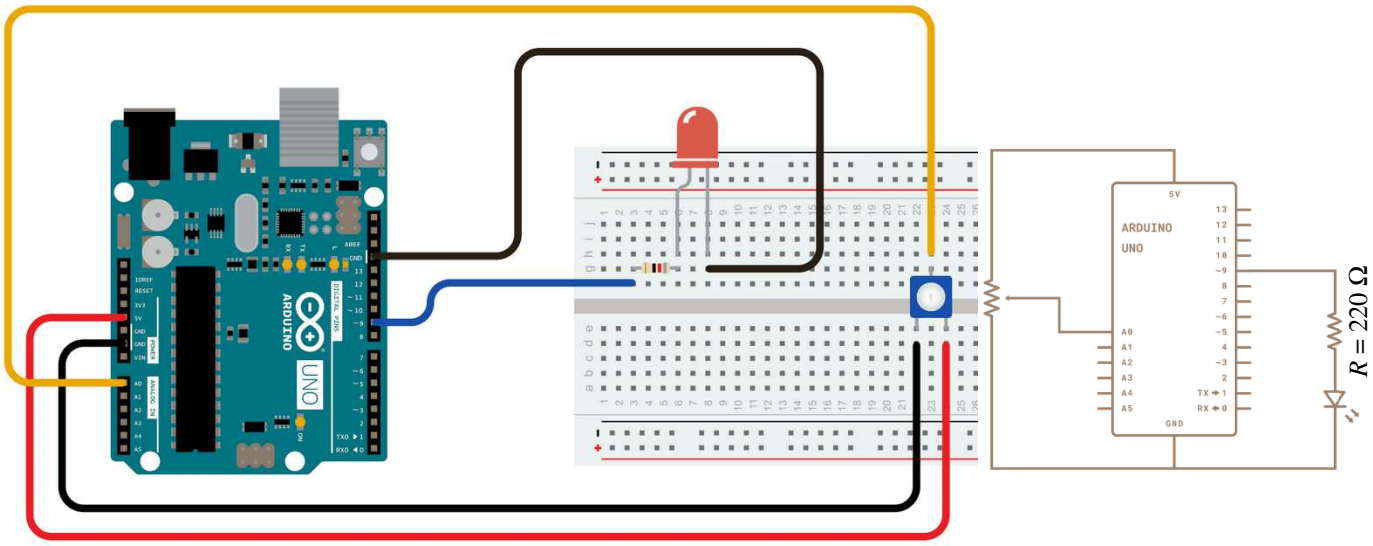
```
1  int led = 9;           // Nom de variable : n° de la broche PWM connectée à la LED
2  int intensite = 0;     // Niveau de luminosité
3  int variationI = 5;    // Pas = ΔI de variation de l'intensité
4
5  void setup() {
6    // Configure la broche 9 = led en sortie (OUTPUT)
7    pinMode(led, OUTPUT);
8  }
9
10 void loop() {
11   // Fixe le niveau de luminosité de la led en écrivant ce niveau à la broche 9
12   analogWrite(led, intensite);
13
14   // Modifie la luminosité à chaque boucle
15   intensite = intensite + variationI;
16
17   // Modifie le sens de variation lorsque les bornes sont atteintes
18   if (intensite <= 0 || intensite >= 255) {
19     variationI = -variationI;
20   }
21   delay(30);
22 }
```

https://docs.arduino.cc/built-in-examples/analog/AnalogInOutSerial/

Application : générer un signal pseudo-analogique à partir d'un autre signal analogique

Cet exemple effectue une synthèse des exemples 3 et 5 et reprend le même matériel.
Le potentiomètre 10 kΩ (simulant ici un capteur) va permettre de fixer le niveau de luminosité de la LED.

`analogRead()` renvoie une valeur dans l'intervalle [0, 1023].
`analogWrite()` accepte des valeurs dans l'intervalle [0, 255].
La fonction **`map(`***`brocheEntréeAnalogique`***`, 0, 1023, 0, 255)`** permet de convertir les valeurs issues d'une broche d'entrée analogique renvoyant des valeurs comprises entre 0 et 1023 en valeurs « compressées » dans l'intervalle 0-255.



Sketch

```
const int analogInPin = A0;  // Entrée broche analogique connectée au potentiomètre
const int analogOutPin = 9;  // Sortie broche PWM analogique connectée à la LED

int sensorValue = 0;         // Valeur lue sur la broche du potentiomètre
int outputValue = 0;         // Valeur écrite sur la broche de la LED

void setup() {
  Serial.begin(115200);
  while (!Serial) {
    delay(100);
  }
}

void loop() {
  sensorValue = analogRead(analogInPin);            // Lecture broche entrée
(capteur)

  // Définition des intervalles de variations
  outputValue = map(sensorValue, 0, 1023, 0, 255);

  analogWrite(analogOutPin, outputValue);           // Ecriture broche sortie (LED)

  // Affichage dans le moniteur série
  Serial.print("Capteur = ");
  Serial.print(sensorValue);
  Serial.print("\t sortie = ");
  Serial.println(outputValue);

  delay(300);
}
```

Principe : enregistrer un signal analogique point par point avec entrées de données au clavier.

Le sketch ci-dessous permet de réaliser l'acquisition de grandeurs (analogiques ou numériques) en contrôlant l'acquisition au clavier :

- soit pour entrer des données au clavier pour une mesure difficile à effectuer via un capteur ;
- soit pour interrompre l'acquisition et effectuer un réglage entre deux mesures.

Par ailleurs, les acquisitions et les sorties sont regroupées dans deux fonctions (non indispensable mais données à titre d'exemple afin d'illustrer la structure de programmes complexes).

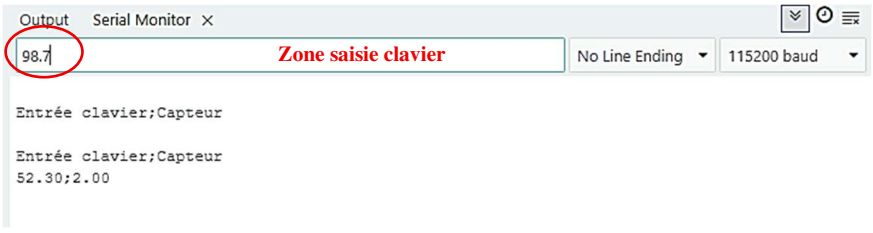
Sketch

```
1  // Un seul capteur dans cet exemple : entrée analogique A0
2  #define capteurPin A0    // Broche Arduino utilisée
3
4  float capteurValeur;     // Variable de stockage de la valeur capteur
5  float entreeClavier;     // Données clavier issues du port série
6
7  void setup() {
8    Serial.begin(115200);
9    while (!Serial) {;}
10    Serial.println("\nEntrée clavier;Capteur"); // Colonnes (en-tête fichier)
11  }
12
13 void loop() {
14   if (Serial.available() > 0) {              // Si données dans le port série
15     entreeClavier = Serial.parseFloat();     // Conversion données port série
16     capteurValeur = lecture(capteurPin);     // Lecture capteur
17     ecriture(entreeClavier, capteurValeur);  // Ecriture dans le port série
18   }
19 }
20
21 float lecture(int broche) {
22   float valeurMesuree = analogRead(broche) / 1023.0 * 5.0; // Conversion
23   delay(2); // delai (ms) pour laisser le CAN réagir
24   return valeurMesuree;
25 }
26
27 void ecriture(float c, float v) {
28   Serial.print(c);
29   Serial.print(";");
30   Serial.println(v);
31 }
```

A la ligne 2, `#define` est une ***directive de compilation*** (cf. page 4).

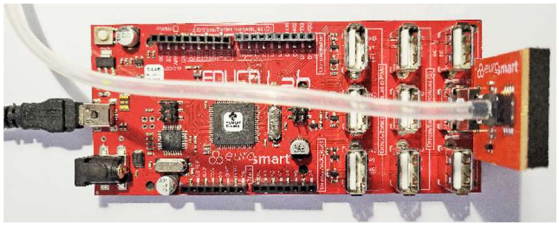
A la ligne 21, on définit une fonction *lecture* admettant deux paramètres et renvoyant une valeur (syntaxe avec type sans *void*).
A la ligne 27, on définit une fonction *ecriture* admettant deux paramètres et ne renvoyant aucune valeur (*void* sans type).
L'ordre d'écriture des fonctions est sans importance.

A la ligne 15, la boucle est interrompue tant que rien n'est entré au clavier dans la zone de saisie (copie d'écran ci-dessous).
Après validation, les lignes 16 et 17 sont exécutées et la boucle recommence.
Lignes 15 et 28 des conversions sont effectuées car les données transitant par le port série sont des chaînes (lecture et écriture).



Application : mesure de pression (capteur Elab-PA) et de volume (clavier).

Carte Educaduino (Arduino Mega) avec capteur de pression absolue :



Le volume est lu sur la seringue et la valeur sera entrée au clavier.

Le capteur de pression absolue Elab-PA renvoie sur la broche **A9** une valeur de type float codée sur 10 bits donc comprise dans l'intervalle [0.0, 1023.0] correspondant à une pression (en Pa) dans l'intervalle [20 000, 400 000] (i.e. entre 200 et 4000 hPa).
Il faut donc définir une fonction permettant de calculer la pression réelle P à partir de la valeur mesurée V :

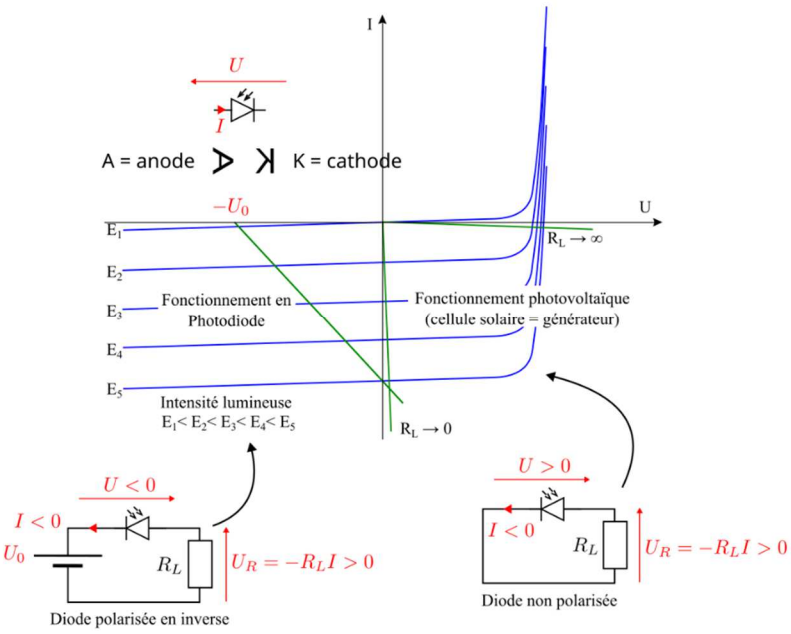
$P = aV + b$ où $a = \dfrac{P_{\max} - P_{\min}}{V_{\max} - V_{\min}}$ et $b = P_{\max} - aV_{\max}$ avec $P_{max} = 400000$, $P_{min} = 20000$, $V_{max} = 1023.0$, $V_{min} = 0.0$.

Ecrire le croquis (ou esquisse) permettant d'entrer les valeurs de volume au clavier et de mesurer la pression avec le capteur et d'afficher les données dans le moniteur série.
Puis récupérer ces données via python, tracer le produit PV en fonction de V. La loi de Mariotte est-elle vérifiée ?

Exemple 8 – Niveaux d'intensité lumineuse – Utilisation d'une photodiode (BPW 34)

Fonctionnement d'une photodiode

En bleu, les caractéristiques $I(U)$ de la photodiode (cf. convention récepteur sur le graphe) pour différentes valeurs E_i de l'éclairement (ou intensité lumineuse).
En vert les caractéristiques du dipôle connecté à la photodiode (montage indiqué pour les deux quadrants inférieurs).



On note R_L la résistance dite de charge (*Load* en anglais).

Dans le cadran supérieur droit, la photodiode se comporte comme une diode « simple ».

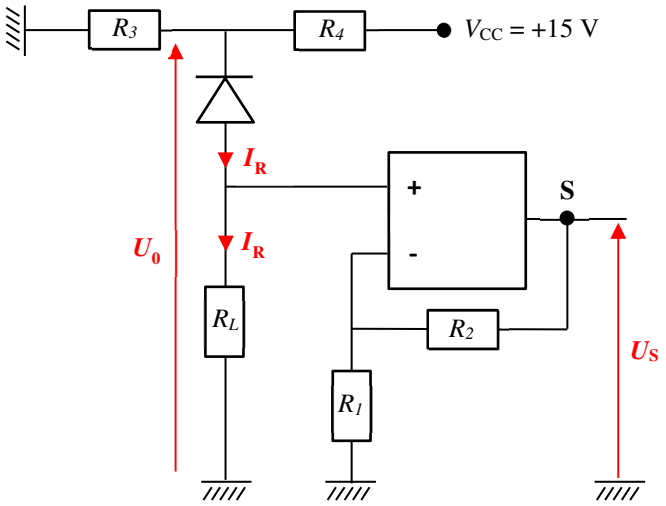
On distingue deux modes de fonctionnement utiles :

- ✓ en cellule solaire (générateur), la diode non polarisée ($V_A > V_K$ donc $U > 0$) débite dans la résistance de charge R_L qui effectue la conversion courant / tension (les deux droites en vert tracées dans ce cadran ont pour équation $I = -\dfrac{U_R}{R_L}$ avec $U_R = U$, cf. schéma) ;
- ✓ en mode photodiode, la diode est polarisée en inverse ($V_K > V_A$ donc $U < 0$) grâce au générateur de f.e.m. U_0 (intensité I est de l'ordre du µA), ***l'intensité électrique I est alors proportionnelle à l'éclairement lumineux E*** (le dipôle connecté aux bornes de la photodiode a pour caractéristique $I = -\dfrac{U + U_0}{R_L}$).

Montage pratique en mode photodiode

On démontre facilement que $U_S = R_L\left(1 + \dfrac{R_2}{R_1}\right) I_R$.

Ordres de grandeur :
$R_1 = 1$ kΩ, $R_2 = 22$ kΩ (amplification)
$R_3 = 1$ kΩ, $R_4 = 10$ kΩ (pont permettant de fixer U_0)
$R_L = 1$ à 10 kΩ



⚠ Avec Arduino, ajuster R_2 et/ou R_L de façon à ce que $U_S < 5$ V !
Valeurs optimales lorsque le bruit n'est pas gênant (R_L assez grand) et U_S aussi grand que possible pour l'éclairement le plus fort mais inférieur à 5 V ***impérativement*** pour mesurer U_S avec Arduino.

Application : vérification de la linéarité ($I \propto E$)

Imaginer un montage avec deux LED permettant de vérifier la linéarité du montage (sans Arduino dans un premier temps).
Etablir un protocole précis et faire un schéma du dispositif.