Tutoriels Python - Arduino

Version en ligne avec liens vers les fichiers individuels : https://gillesbeharelle.fr/deuxColonnes/table_liens_python_arduino

Arduino – Découvrir en expérimentant

Arduino – Exemples

Ecrire dans le port série
`Serial.begin(v), Serial.print(s), Serial.println(s), delay(n)`

Lire des données dans le port série avec Python
`millis()`

Acquisition d'un flux de données analogiques
`analogRead(pin)`

Acquisition de données numériques
`pinMode(pin, mode), digitalRead(pin)`

Sortie numérique PWM simulant un signal analogique
`analogWrite(pin, niveau)`

Sortie analogique commandée par une entrée analogique
`map(pin, a, b, x, y)`

Acquisition de données analogiques et entrées clavier
`Fonctions, #define, Serial.parseFloat()`

Moteur de Stirling – Tracé d'un cycle (*P*, *V*)
`Tests, opérateurs logiques`

Arduino – Mémento

Arduino – Mémento

Microcontrôleurs
Carte Arduino Uno - Brochage
Carte Arduino Uno - Caractéristiques techniques
Arduino IDE – Editeur – Moniteur série
Connexion de la carte à un ordinateur

C++ Arduino - Mémento

Mémento – Langage C++

void, setup(), loop()
Syntaxe
Types, opérateurs
Boucles, instructions conditionnelles
Tableaux
Port série, Serial.begin(v), Serial.print(s), Serial.println(s)

Python – Fichiers texte

Structure des fichiers texte
Principe de l'extraction de données dans un fichier texte
Ouverture d'un fichier : modes lecture, écriture ou ajout
Lecture d'un fichier
Lecture d'un fichier – Traitement des données et stockage

Python – Acquisition de données via le port série

Port série
Code minimal - Test du port série et visualisation des données dans le shell python
Détection automatique du port série
Traitement des données – Stockage des données dans une liste puis un tableau numpy

Ecrire dans le port série

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

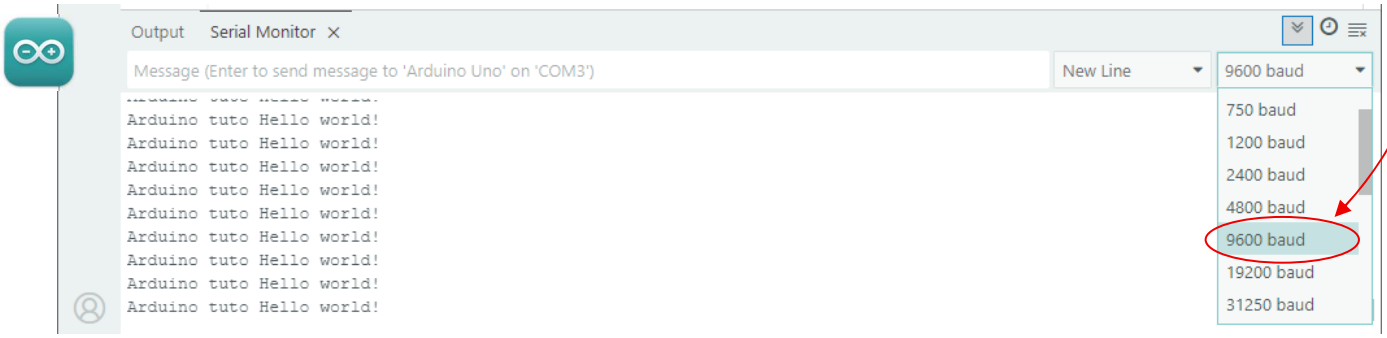
Le sketch suivant illustre la communication entre la carte Arduino et le ***port série*** de l'ordinateur (physiquement, via le câble USB).

Instructions

`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**.



Remarque : la boucle étant infinie, le flux dans le port série est ininterrompu.

Que signifie l'instruction **`!Serial`** (ligne 9) ?
Quelle est la différence entre les instructions **`print()`** et **`println()`** ? Ne pas hésiter à modifier le code pour effectuer des tests.

Lire des données dans le port série avec Python

Prérequis : Ecrire dans le port série

Application : récupérer via python un flux de données, les visualiser en cours d'acquisition et les sauvegarder

Instructions

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

Ecrire un programme (dans l'IDE Arduino) permettant de ***simuler*** un flux de données dans le port série en respectant le cahier des charges ci-dessous. La boucle simulant les mesures est interrompue lorsqu'un temps maximum (défini dans le sketch) est atteint.

Le programme doit :

- définir une variable `tMax = 5000` représentant la durée totale du processus de « mesure » ;
- initialiser une variable `t0` en capturant le temps au début de la boucle `loop()` (cf. instruction `millis()` ci-dessus) ;
- initialiser un compteur de mesures nommé `cpt` au début de la boucle `loop()` ;
- initialiser une variable `ti = t0` représentant l'instant d'acquisition de la mesure n° *i* ;
- exécuter en boucle les instructions suivantes tant que la durée du processus n'excède pas le temps maximal :
  - attendre 100 ms ;
  - incrémenter le compteur de mesures ;
  - capturer la nouvelle valeur de `ti` ;
  - calculer le temps écoulé `t` depuis t_0 en s ;
  - calculer une expression quelconque à partir de ce temps (relation affine par exemple) afin de simuler la mesure d'une grandeur au cours du temps (utiliser une variable nommée `mesure_capteur`) ;
  - écrire les valeurs de `t`, `mesure_capteur` et `cpt` dans le port série (séparés par des points-virgules).
- écrire `"Stop"` dans le port série une fois la boucle précédente achevée.

Programme Python (tutoriels détaillés : documents « Python – Fichiers texte » et « Python – Acquisition de données via le port série »)

```
1  import serial
2
3  # Communication
4  port_serie = 'COM3'           # Cf. Arduino IDE (port sélectionné)
5  bauds = 115200                # Cf. sketch arduino : Serial.begin(bauds)
6  # Enregistrement des mesures
7  dossier = ""                  # Dossier courant (dossier de ce fichier python)
8  nom_fichier = "exemple2.txt"  # A personnaliser - Préférer un chemin ABSOLU
9  chemin = dossier + nom_fichier
10
11 ps = serial.Serial(port_serie, bauds) # Ouverture du port série
12 fichier = open(chemin, "w+")          # Ouverture du fichier en écriture
13 while True:                           # Boucle "infinie"
14     ligne = ps.readline()             # Lecture d'une ligne sur le port série
15     ligne = ligne.decode("utf-8")     # readline -> binaire, conversion
16     ligne = ligne.strip('\n')         # Traitement (suppression \n fin de ligne)
17     if 'Stop' in ligne:               # Message d'arrêt défini dans sketch Arduino
18         break                         # Sortie de la boucle while
19     print(ligne)                      # Vérification visuelle dans le shell
20     fichier.write(ligne)              # Ecriture de la ligne dans le fichier
21 fichier.close()                       # Fermeture du fichier
22 ps.close()                           # Fermeture du port série
```

Protocole

1. Téléverser le sketch sur la carte Arduino.
2. ***Fermer le moniteur série Arduino*** (ne pas déconnecter la carte du port USB).
3. ***Vérifier le paramétrage lignes 4 (port) et 5 (vitesse de transmission)*** dans le programme python.
4. Exécuter le programme python : les données lues doivent s'afficher dans le shell.
5. Vérifier le contenu du fichier en l'ouvrant (possible depuis pyzo).

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)

Il existe de nombreuses façons d'interrompre la lecture dans le programme python (durée d'acquisition, nombre de mesures…) qui ne nécessitent pas d'inclure dans le sketch Arduino l'envoi d'une instruction particulière.

Acquisition d'un flux de données analogiques - analogRead()

Application : enregistrer un flux ininterrompu de données analogiques

Instructions

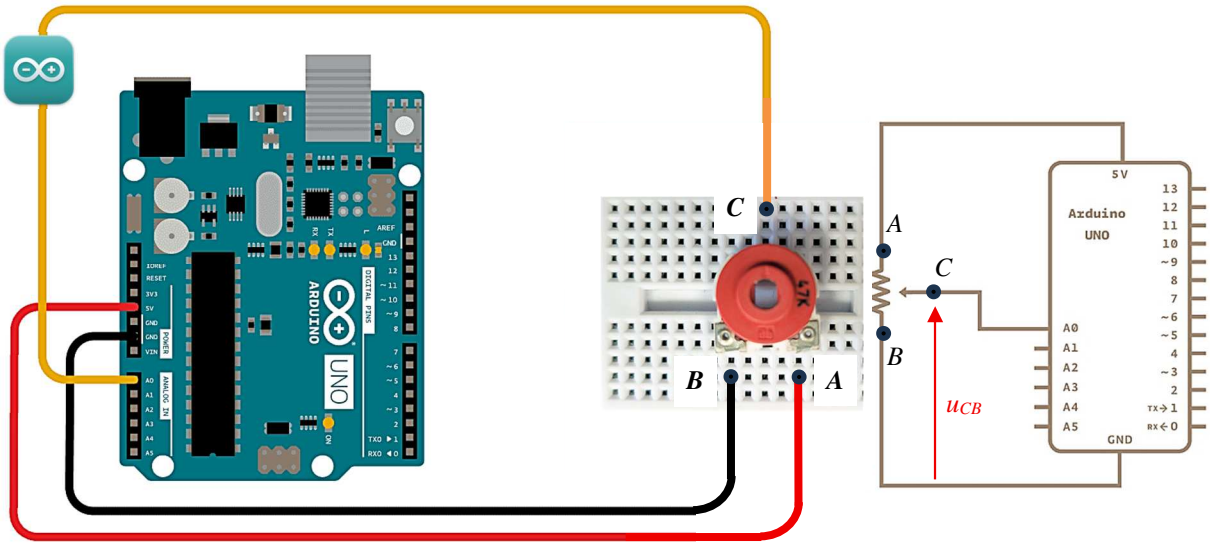
`analogRead(`*`pin`***`)`** Lire la valeur binaire sur la broche *pin* (A0 à A5), cf. brochage de la carte utilisée.

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.

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 (47 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 =
17
18   Serial.println(U_Volts);
19
20   delay(500);  // Utile pour ralentir le flux de données
21 }
```

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

Visualiser les variations de U dans le « Serial plotter » .

Acquisition de données numériques - digitalRead()

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

Application : enregistrer un flux ininterrompu de données numériques

Instructions

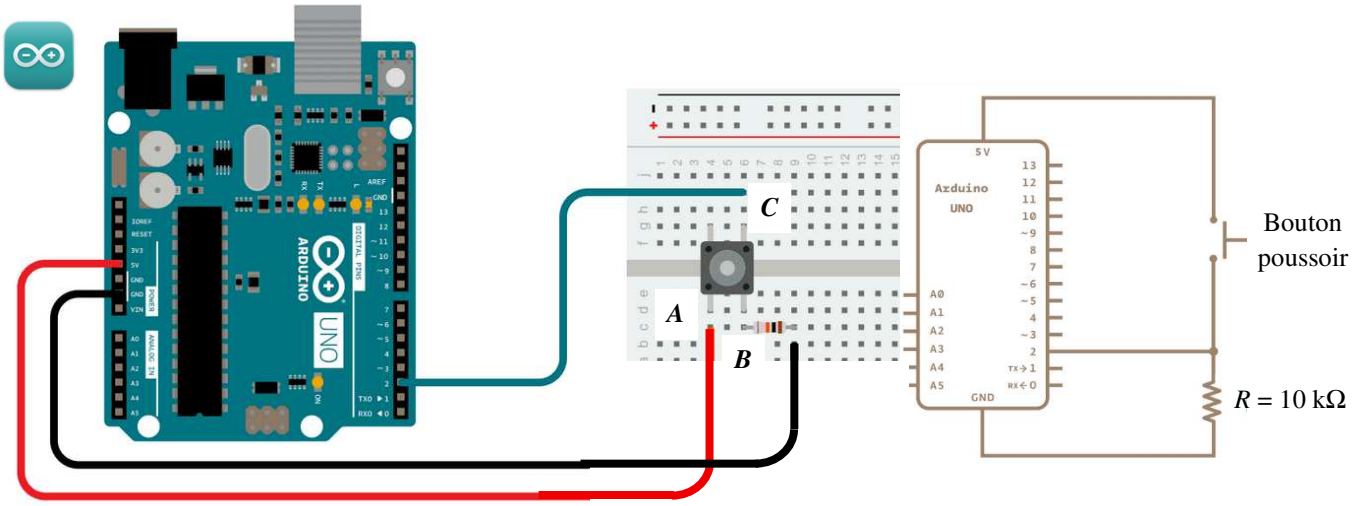
pinMode(*pin, mode***)**	Configurer la broche *pin* en entrée (*mode* = INPUT) ou en sortie (*mode* = OUTPUT).
digitalRead(*pin***)**	Lire la valeur booléenne sur la broche *pin* (2, 4, 7, 8, 12, 13).

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.

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 }
```

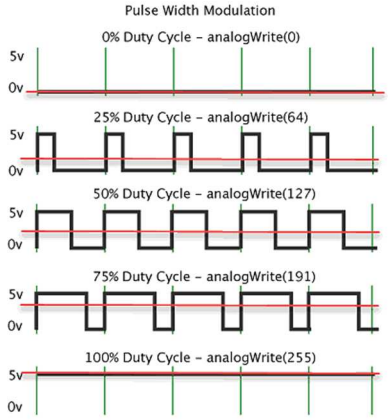
Sortie numérique PWM simulant un signal analogique - analogWrite()

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

Application : générer un signal pseudo-analogique

Instructions

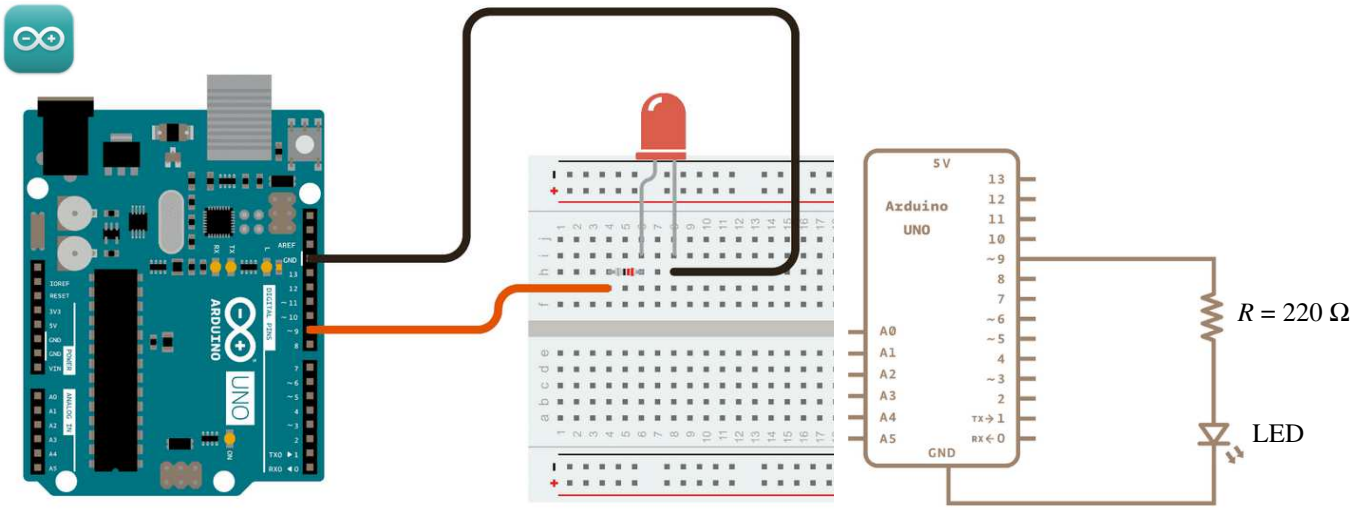
`analogWrite(`*`pin, niveau`***`)`** Ecrire la valeur *niveau* sur la broche *pin* (~3, ~5, ~6, ~9, ~10, ~11 PWM = symbole ~).
`analogWrite()` accepte des valeurs entre 0 et 255.



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.

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   //
15   intensite = intensite + variationI;
16
17   //
18   if (intensite <= 0 || intensite >= 255) {
19     variationI = -variationI;
20   }
21   delay(30);
22 }
```

Expliquer le code (lignes 15, 18 et 19).

Sortie analogique commandée par une entrée analogique

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

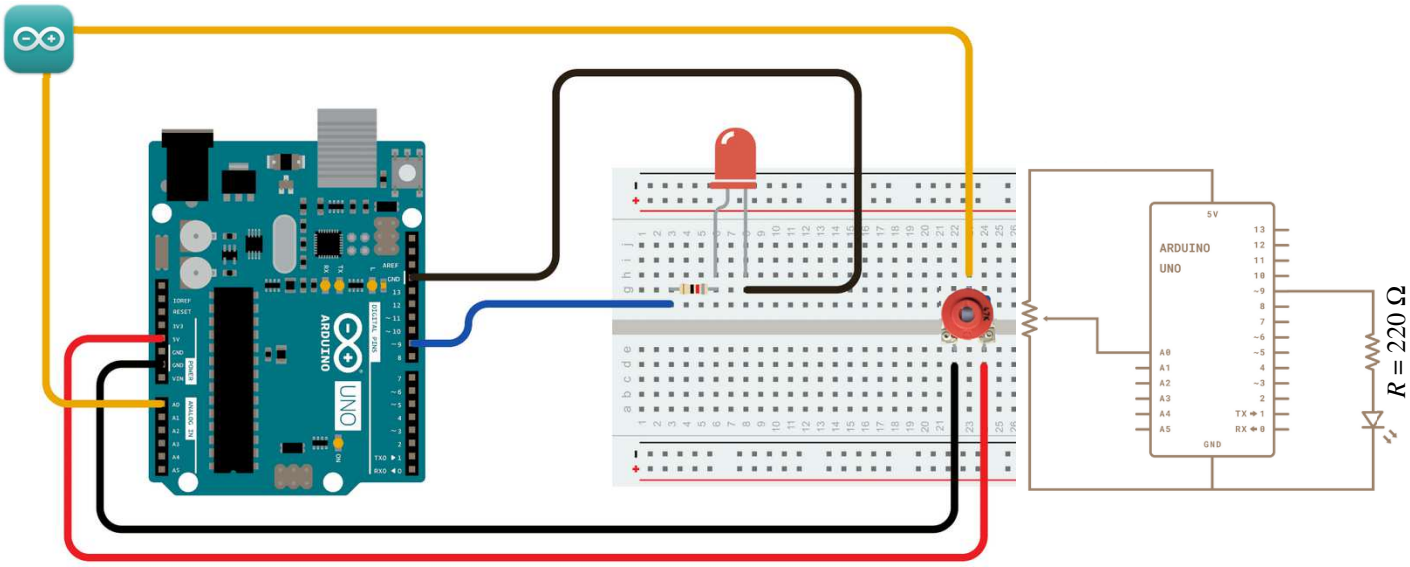
Instructions

`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 dans l'intervalle 0-255.

Cet exemple effectue une synthèse des exemples 3 et 5 et reprend le même matériel.
Le potentiomètre 47 kΩ (simulant ici un capteur) va permettre de fixer le niveau de luminosité de la LED.



Ecrire le code permettant de faire varier la luminosité de la LED en agissant sur le potentiomètre.
Les valeurs lues sur la broche d'entrée et écrites sur la broche de sortie seront affichées dans le moniteur série (vérifier que les intervalles de variation sont corrects).

Acquisition de données analogiques et entrées clavier

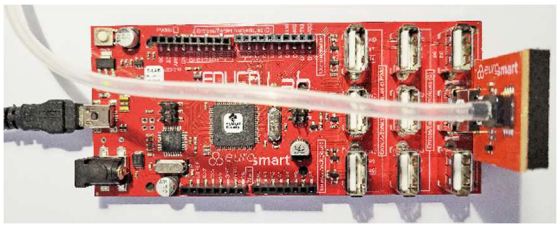
Principe : enregistrer un signal analogique point par point avec entrées de données au clavier.

Instructions

`void``([paramètres])`	Fonction ne renvoyant rien (avec ou sans paramètres)
type nom_fonction `([paramètres])`	Fonction renvoyant en résultat (**`return`** *valeur*)
`Serial.parseFloat()`	Renvoie un flottant lu dans le port série

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

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

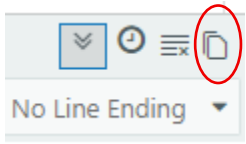


Le volume est lu sur la seringue et la valeur sera entrée au clavier.

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, 1023] 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 ou sketch) 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.



Exploitation rapide : copier les données directement dans le moniteur série à l'aide du bouton :
Coller dans un fichier texte (Notepad++, Bloc-notes) et enregistrer au format texte (.txt).
Exploiter dans Régressi : tracer le produit PV en fonction de V. La loi de Mariotte est-elle vérifiée ?

Sauvegarde des données dans un fichier : cf. « Lire des données dans le port série avec Python ».

Aide

Le sketch page suivante 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 (principe → code à adapter)

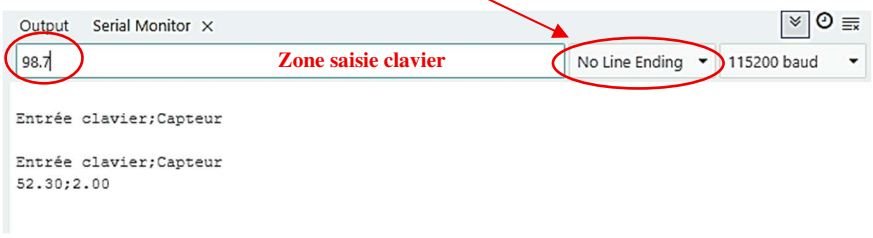
```
1  // Un seul capteur dans cet exemple : entrée analogique A0
2  #define capteurPin A0    // Broche Arduino utilisée
3
4  float capteurValeur;     // Variable de stockage de la valeur capteur
5  float entreeClavier;     // Données clavier issues du port série
6
7  void setup() {
8    Serial.begin(115200);
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. « Arduino – Mémento »).

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).

⚠ Bien sélectionner 'No Line Ending' dans le moniteur série.



Sans cette précaution, deux lignes apparaissent à chaque saisie.

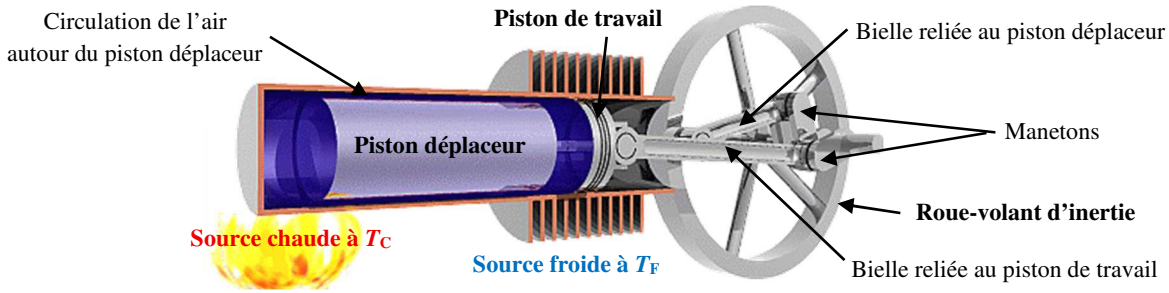
Moteur de Stirling – Tracé d'un cycle (P, V)

Principe du moteur Stirling (Robert et James Stirling 1816)

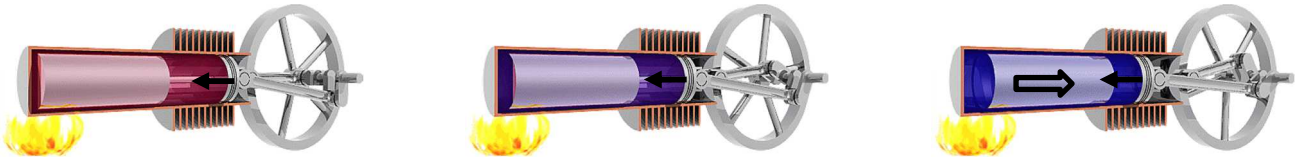
Le moteur Stirling est un moteur à ***combustion externe*** (toute source de chaleur peut être utilisée : énergie solaire, énergie géothermique, énergie nucléaire…) et à fluide de travail en ***cycle fermé*** (l'air interne au moteur ne sort jamais de celui-ci).
Les moteurs thermiques usuels sont à *combustion interne* en *cycle ouvert*.
La spécificité de ce moteur réside dans un ***régénérateur*** (échangeur thermique interne) qui améliore son efficacité.

Moteur Stirling de type bêta

Le moteur comporte deux pistons, un ***piston de travail ou piston moteur*** et un ***piston « déplaceur »*** reliés à la ***roue d'inertie*** via deux bielles fixées à la roue grâce à deux manetons décentrés par rapport à l'axe de rotation de la roue (cf. schéma ci-dessous).
Cet embiellage crée pour le piston moteur un retard de phase de $\pi/2$ (un quart de tour) par rapport au piston déplaceur.
Le piston déplaceur (de plus petit diamètre que le cylindre) possède un double rôle :
- faire passer alternativement le gaz de la source chaude à la source froide (ailettes de refroidissement dans l'air ambiant) ;
- contribuer à réchauffer ou refroidir l'air au cours de ses transformations.

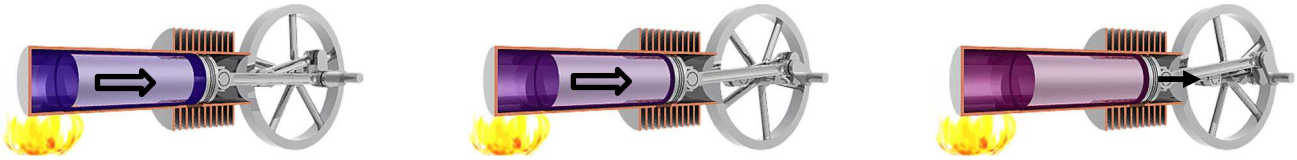


Bien distinguer l'air du côté piston moteur-source froide de l'air du côté source chaude pour analyser le fonctionnement.



Piston déplaceur en position basse

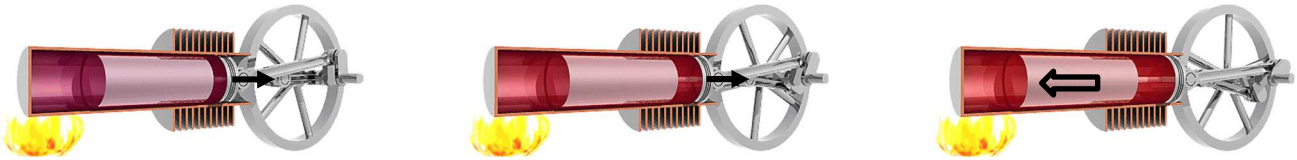
La totalité de l'air, encore chaud, est au contact de la source froide : sa température baisse ainsi que sa pression. Puis il commence à être refoulé vers la source chaude par les deux pistons qui se rapprochent tandis que le piston déplaceur se refroidit au contact de la source froide.
Le volume entre les deux pistons diminue : la compression du volume « moteur » s'effectue au contact du déplaceur refroidi, la ***compression*** est ***isotherme*** ($V\downarrow$, $P\uparrow$ à T = constante).



Point mort bas

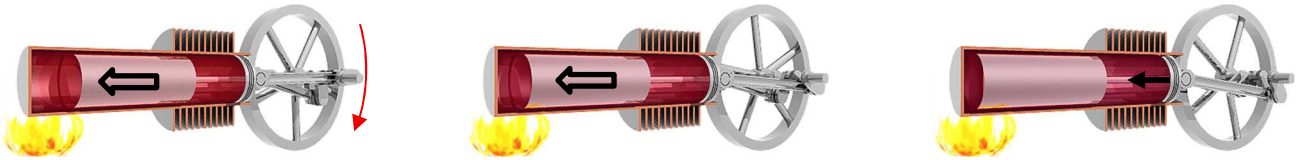
Piston de travail en position basse

Au voisinage du point mort bas, le volume « moteur » varie peu autour de son minimum : la ***compression*** est ***isochore***. L'air est progressivement transféré du côté de la source chaude par le piston déplaceur qui se rapproche de sa position la plus haute.



Piston déplaceur en position haute

L'air réchauffé dont la pression a augmenté commence à repousser le piston moteur tandis que le déplaceur contribue à le transférer vers le piston moteur réchauffant au passage le piston déplaceur. Les pistons s'éloignent, le piston moteur est refoulé par l'air chaud dont la pression va diminuer. Le piston déplaceur est alors au contact de la source chaude : la phase motrice est une ***détente isotherme*** ($V\uparrow$, $P\downarrow$ à T = constante).



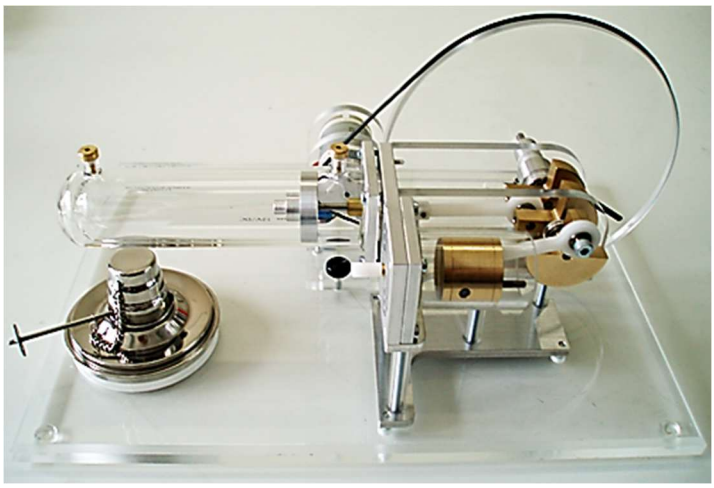
Point mort haut (PMH)

Piston de travail en position haute

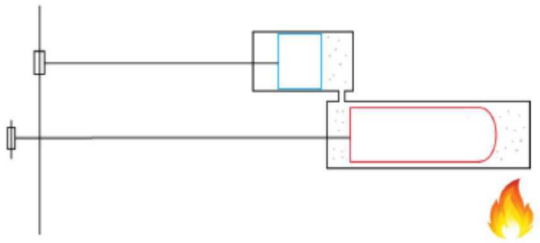
Au voisinage du point mort haut, le volume « moteur » varie peu autour de son maximum et l'air du côté « moteur » est au contact de la source froide, il se refroidit et sa pression diminue : la détente est isochore. Le piston déplaceur transfère progressivement l'air vers la source froide.

Moteur Stirling de type gamma

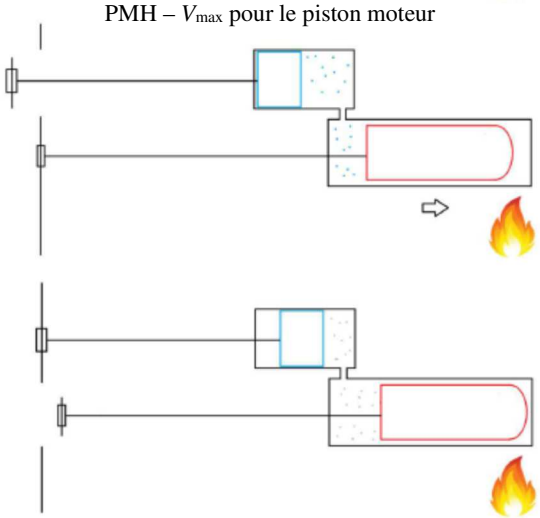
Dans un moteur de type gamma, le fonctionnement est plus facile à appréhender car les deux pistons évoluent dans deux cylindres séparés mais en communication l'un avec l'autre (schéma ci-contre et photo ci-dessous).



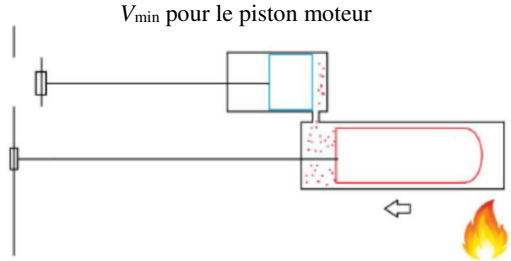
Dans ce moteur, le régénérateur est la plaque en métal entre les deux cylindres.



PMH – V_{max} pour le piston moteur



V_{min} pour le piston moteur



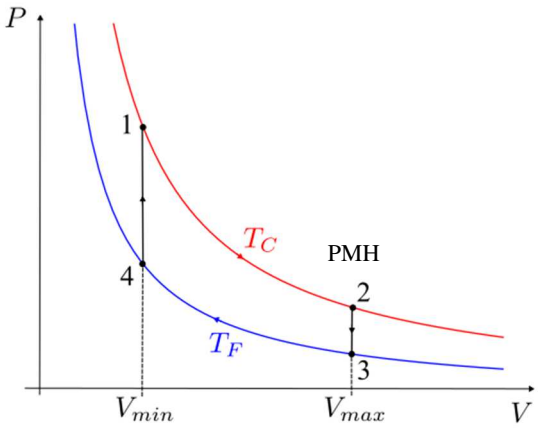
Modélisation du cycle de Stirling

On note a le taux de compression défini par : $a = \dfrac{V_{max}}{V_{min}}$.

On suppose que l'air est un gaz parfait.

En l'absence de régénérateur, le rendement de ce moteur est :

$$\eta_{sr} = 1 - \frac{T_F \ln a + \dfrac{T_C - T_F}{\gamma - 1}}{T_C \ln a + \dfrac{T_C - T_F}{\gamma - 1}}.$$



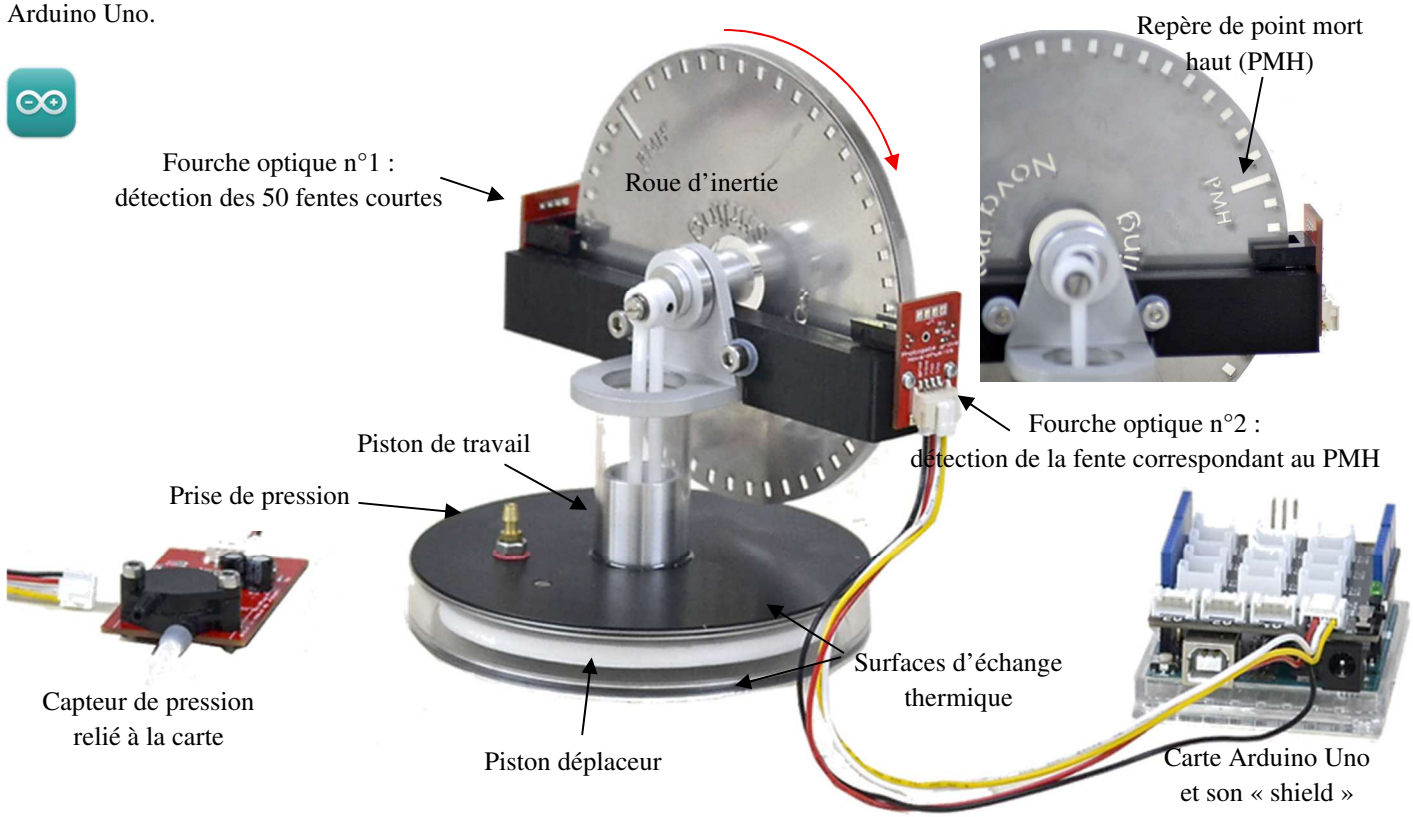
En présence d'un régénérateur parfait, on suppose que celui-ci récupère entièrement l'énergie nécessaire au réchauffage isochore (de 4 à 1) au cours du refroidissement isochore (de 2 à 3) : le piston déplaceur est effectivement au voisinage de la source chaude de 2 à 4 de façon à restituer l'énergie emmagasinée de 4 à 2.

L'efficacité devient alors : $\eta_{ar} = 1 - \dfrac{T_F}{T_C}$.

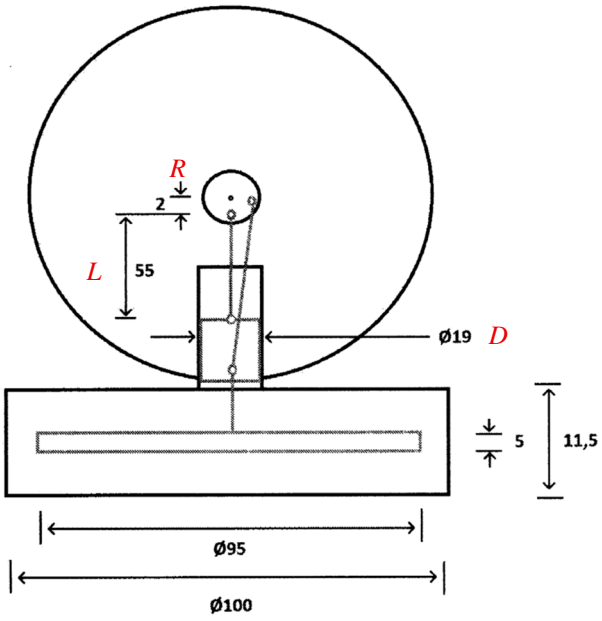
On reconnaît l'efficacité de Carnot valable pour un cycle ditherme réversible appelé cycle de Carnot et constitué de deux isothermes et de deux isentropiques. Cependant le cycle de Stirling n'est pas un cycle de Carnot car constitué de deux isothermes et de deux isochores et n'est pas ditherme en raison des échanges avec le régénérateur.
Néanmoins ce résultat permet de comprendre que le rendement est amélioré en présence du régénérateur (qui ne saurait être parfait en pratique).

Moteur Stirling instrumenté

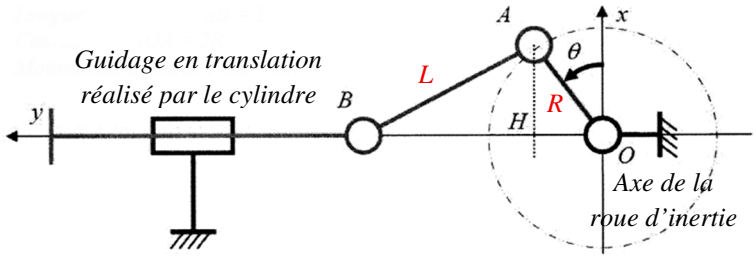
Deux fourches optiques permettant de déterminer la position de la roue d'inertie et un capteur de pression sont reliés à une carte Arduino Uno.



Données techniques



Distances / longueurs en mm



Mouvement d'entrée : $\theta(t) = \omega t$ (sens + = sens horaire de rotation)

L'angle θ est connu grâce aux deux fourches optiques (cf. ci-dessous).

On en déduit : $\theta = \dfrac{\pi}{2} + i \times \delta\theta$ (observer le passage du PMH la fourche 2)

Le volume du cylindre moteur est alors :

$$V_{cyl} = \left[OB - (L-R)\right]\frac{\pi D^2}{4} = \left[R\sin\theta + \sqrt{L^2 - R^2\cos^2\theta} - (L-R)\right]\frac{\pi D^2}{4}$$

Longueur de la bielle du piston de travail : $L = 55$ mm

Ecart OA du maneton de bielle moteur à l'axe de la roue d'inertie : $R = 2$ mm

Diamètre du piston de travail : $D = 19$ mm

Nombre de fentes découpées dans la roue d'inertie : $N = 50 \Rightarrow$ Angle entre deux fentes successives : $\delta\theta = 2\pi / N$

Parmi ces fentes, une est plus longue et permet de détecter le passage par le point mort haut (PMH) (position du piston de travail la plus haute donc volume maximum). Cette fente est détectée par la fourche optique n°2 (et par la fourche n°1).

A chaque passage d'une fente devant la fourche optique n°1, une mesure de pression est enregistrée.

Au cours d'un tour de la roue d'inertie, on dispose de 50 triplets de la forme $(i, k_{PMH}, \delta P_{discr})$ où :

- i est le numéro de la fente comptée à partir du PMH (celui-ci correspond à $i = 1$) ;
- k_{PMH} est le nombre de tours effectués par la roue (i.e. le nombre de passage de la fente la plus longue devant la fourche 2) ;
- δP_{discr} est la pression différentielle (différence entre la pression atmosphérique et la pression dans le cylindre).

Une surpression nulle correspond à 2,5 V pour une alimentation entre 0 et 5V.

La tension de sortie $U_{capteur}$ est discrétisée par le CAN 10 bits de la carte Arduino et fournit la valeur δP_{discr}.

La valeur de la pression absolue en **Pa** est alors : $P = P_0 + \delta P = P_0 + \frac{P_{discr} - 512}{1023} \times 5 / Sb$ où Sb est la sensibilité du capteur.

$Sb = \frac{\delta U_{capteur}}{\delta P_{réelle}} = 1{,}03$ V/**k**Pa = 1,03/1000 V/Pa.

Par ailleurs, le volume minimal du cylindre moteur est $V_{min} = 55\ 660$ mm^3 hors volume mort (volume ajouté par la prise de pression).

Le volume réel du cylindre est donc : $V = V_{min} + \left[R \sin\theta + \sqrt{L^2 - R^2 \cos^2\theta} - (L - R) \right] \frac{\pi D^2}{4}$.

Sketch

Hypothèse : la fréquence des mesures est supérieure à la fréquence de rotation, i.e. tous les changements d'état des fourches optiques sont détectés.

Le programme ne commence à écrire les données qu'à partir d'une détection du PMH de façon à disposer des valeurs correspondant à un cycle entier et on définit par ailleurs un nombre maximum de cycles à enregistrer.

```
1  #define capteurPression A0  // Capteur de pression sur la broche A0
2  #define fourche1 A1         // Fourche optique n°1 sur la broche A1
3  #define fourche2 A2         // Fourche optique n°2 sur la broche A2 (détection PMH)
4  #define sepCol ";"          // Séparateur de colonnes -> identique dans pgm python
5
6  int compteur1 = 0;     // Compteur fourche optique n°1 (permet de calculer le volume)
7  int compteur2 = 0;     // Compteur fourche optique n°2 (nombre de passages par le PMH)
8  int etatFourche1;      // Etat de la fourche optique n°1
9  int etatFourche2;      // Etat de la fourche optique n°2
10 int memoire1 = LOW;    // Mémoire de l'état de la fourche optique n°1
11 int memoire2 = LOW;    // Mémoire de l'état de la fourche optique n°2
12 unsigned long t;       // Instant de mesure
13 int cpt2Max = 11;      // Nbre de tours à enregistrer (nombre de lignes à lire par python)
14
15 void setup() {
16   Serial.begin(115200);
17   pinMode(fourche1,INPUT);            // Configure la broche de la fourche n°1 en lecture
18   pinMode(fourche2,INPUT);            // Configure la broche de la fourche n°2 en lecture
19   while (!Serial) {;}                 // Ne rien faire tant que port série non disponible
20 }
21
22 void loop() {
23     while (compteur2 < cpt2Max) {
24       // Détection du passage de la fenêtre du PMH devant la fourche optique n°2
25       etatFourche2 = digitalRead(fourche2);   // Lecture de l'état de la fourche n°2
26       // Si état fourche 2 différent de état enregistré ET état "haut" (faisceau détecté)
27       if((etatFourche2 != memoire2) && (etatFourche2 == HIGH)) {
28             compteur2++;       // On incrémente le compteur2 (nbre de passages par le PMH)
29             compteur1 = 0;     // On remet le compteur1 à zero (angle dans [0, 2*pi])
30       }
31       // Détection du passage d'une des 50 fenêtres devant la fourche optique n°1
32       etatFourche1 = digitalRead(fourche1);   // Lecture de l'état de la fourche n°1
33       // Si état fourche 1 différent de état enregistré ET état "haut" (faisceau détecté)
34       // La condition compteur2 > 0 permet de démarrer la transmission sur un PMH
35       if((etatFourche1 != memoire1) && (etatFourche1 == HIGH) && (compteur2 > 0)) {
36         compteur1++;           // On incrémente le compteur1
37         t = micros();          // Instant de mesure en µs
38         // Affichage des quadruplets de valeurs de compteur1, compteur2, pression, t
39         Serial.print(compteur1); Serial.print(sepCol);
40         Serial.print(compteur2); Serial.print(sepCol);
41         Serial.print(analogRead(capteurPression)); Serial.print(sepCol);
42         Serial.println(t);
43       }
44       // Mémorisation des états pour détecter les changements à la lecture suivante
45       memoire1 = etatFourche1;
46       memoire2 = etatFourche2;
47     }
48     Serial.println("Stop");
49     while (true) {;}
50 }
```

⚠ Ligne 11 : pour les tests, utiliser une valeur « raisonnable » pour la variable `cpt2Max` (3 ou 4).
Noter que la numérotation débute à 0 mais que le cycle n°0 sera ignoré car incomplet donc le premier cycle enregistré aura le n°1 et que le dernier aura pour n° cpt2Max-1 (donc 10 cycles complets avec `cpt2Max = 11`) .

Programme Python **Ressources :**

💻 Le sketch Arduino et le programme python à compléter sont disponibles en téléchargement via le QRcode (lien vers la rubrique TP de gilles.beharelle.fr) : Tutoriels Python – Arduino, à la rubrique Arduino – Découvrir en expérimentant : Moteur de Stirling.

✏ Compléter le code de la fonction calcul_PV (directement dans Pyzo ou Spyder)

```python
import serial
import serial.tools.list_ports
import matplotlib.pyplot as plt
from matplotlib import animation
import matplotlib.colors as mcolors
import numpy as np

# calcul_PV permet de tracer :
#    - tous les cycles avec num_cycle=None
#    - un unique cycle avec num_cycle= numéro cycle souhaité
def calcul_PV(dataN, num_cycle=None, Po=1e5, unite='mm3'):
    """ dataN = tableau de lignes de la forme [i, k, Pdiscr, t]
        i = n° de la fente comptée à partir du PMH
        k = nombre de tours (nombre de passages au PMH)
        Pdiscr = pression différentielle renvoyée par CAN Arduino (bits)
        t = instant de mesure (µs)
        num_cycle = n° du cycle souhaité = l'une des valeurs de k dans data
        Po  : pression atmosphérique en Pa
        unite : 'mm3' ou 'm3' ou 'SI'
    """
    assert unite in ['mm3', 'm3', 'SI'], "Unité incorrecte ('mm3', 'm3' ou 'SI)"
    if num_cycle is None:   # Calculs pour tous les cycles
        data = dataN
    else:                   # Calculs pour le cycle spécifié par num_cycle
        assert float(num_cycle) in dataN[:,1], "n° cycle incorrect"
        data = np.array([l for l in dataN if l[1]==float(num_cycle)])

    # Caractéristiques mécaniques du moteur
    L =                         # Longueur bielle piston moteur (mm)
    R =                         # Rayon maneton bielle moteur (mm)
    D =                         # Diamètre piston moteur (mm)
    S =                         # Surface piston moteur (mm**2)
    dtheta =                    # Angle entre 2 fentes (rad)
    Vmin =                      # Volume minimum du cylindre moteur (mm**3)

    # Calcul du volume du gaz dans le cylindre moteur
    i =                         # n° fente à partir du PMH
    theta =                     # Angle correspondant à la fente n°i
    OB =                        # Position piston
    V =                         # Volume (mm**3)
    if unite in ['m3', 'SI']:
        V *= 1e-9

    # Capteur de pression différentielle
    Sb =                        # Sensibilité = dV/dPréelle = 1.03/1000 V/Pa

    # Pression
    dPd =                       # Pression différentielle discrétisée en bits
    dPv =                       # Pression différentielle en V
    dPr =                       # Pression différentielle réelle en Pa
    P =                         # Pression absolue en Pa

    # Instant de mesure
    t =                         # t en µs (cf. sketch Arduino)

    return P, V, t
```

✏ En lisant le code du programme fourni, déterminer les noms des tableaux/listes contenant les données :
- au format texte (tableau 1D de lignes) ;
- au format numpy (tableau 2D).

💡 Des informations supplémentaire concernant la fonction portArduino sont disponibles via le lien précédent (Tutoriels Python – Arduino, à la rubrique Python – Tutoriels détaillés : Acquisition de données via le port série).

💻 La dernière cellule du programme python fourni permet de sauvegarder dans un fichier texte les données volatiles (perdues lors d'un redémarrage du noyau python par exemple).

🛠 Acquisition des données

1. Brancher la carte Arduino sur un port USB du PC (à l'arrière du PC et non en façade).
2. Téléverser le sketch sur la carte, prendre `cpt2Max = 3` .
3. Afficher le moniteur série et faire tourner délicatement le moteur dans le sens indiqué page 3 de ce document : les données apparaissent dans le moniteur série (relancer le moteur si nécessaire tant que l'affichage n'est pas interrompu).
4. Fermer le moniteur série.
5. Identifier la variable en relation avec `cpt2Max` dans le programme python et ajuster sa valeur.
6. Exécuter le programme python et relancer le moteur à la main comme à l'étape 3 : les données sont affichées dans le shell puis un cycle est tracé.
7. Modifier la variable `cpt2Max` du sketch (prendre 11 par exemple) et la variable python correspondante.
8. Poser le moteur sur une tasse d'eau chaude, attendre quelques secondes puis le lancer très doucement comme à l'étape 3 s'il ne démarre pas seul.
9. Exécuter le programme python.
10. ⚠ Lorsque le moteur s'arrête poser le moteur sur un **chiffon** et non sur la ~~paillasse~~.
11. Sauvegarder l'acquisition dans un fichier (effectuer une nouvelle acquisition si nécessaire).
12. Ouvrir le fichier (possible depuis l'explorateur de fichiers de Pyzo) et vérifier que les données sont correctes : pas de lignes tronquées ou sautées, nombre entier de cycles complets (la première colonne doit varier de 1 à 50 de façon cyclique).

✏ Ecrire un programme permettant de lire, traiter les données du fichier de sauvegarde et les sauvegarder au format texte dans une liste nommée dataFichier (cf. Tutoriels Python – Arduino, à la rubrique Python – Tutoriels détaillés : Fichiers texte).
Transformer cette liste en tableau numpy nommé dataFichierN.
Vérifier que la fonction calcul_PV permet de tracer le(s) cycle(s).

✏ Traitement des données – Aspects thermodynamiques

13. Définir « à la main » (en analysant les données d'un cycle) ou écrire une fonction permettant de déterminer les index des volumes minimum et maximum dans la liste des volumes au cours d'un unique cycle.
14. Ecrire une fonction aire(x, y) permettant de calculer l'aire sous la courbe $y(x)$ par la méthode des trapèzes.
15. Ecrire une fonction puissance(numCycle) permettant de déterminer la puissance fournie par le moteur au cours du cycle n°numCycle (cette fonction appellera la fonction calcul_PV de façon à disposer des données pour ce cycle en précisant que l'unité doit être m^3).
16. Tester cette fonction.
17. Ecrire une fonction puissanceMoyenne(tab) permettant d'obtenir la puissance moyenne calculée sur tous les cycles disponibles dans le tableau de données *tab*.

🛠 Prolongements

18. Imaginer un protocole permettant d'estimer le rendement de ce moteur.

Arduino - Mémento

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.

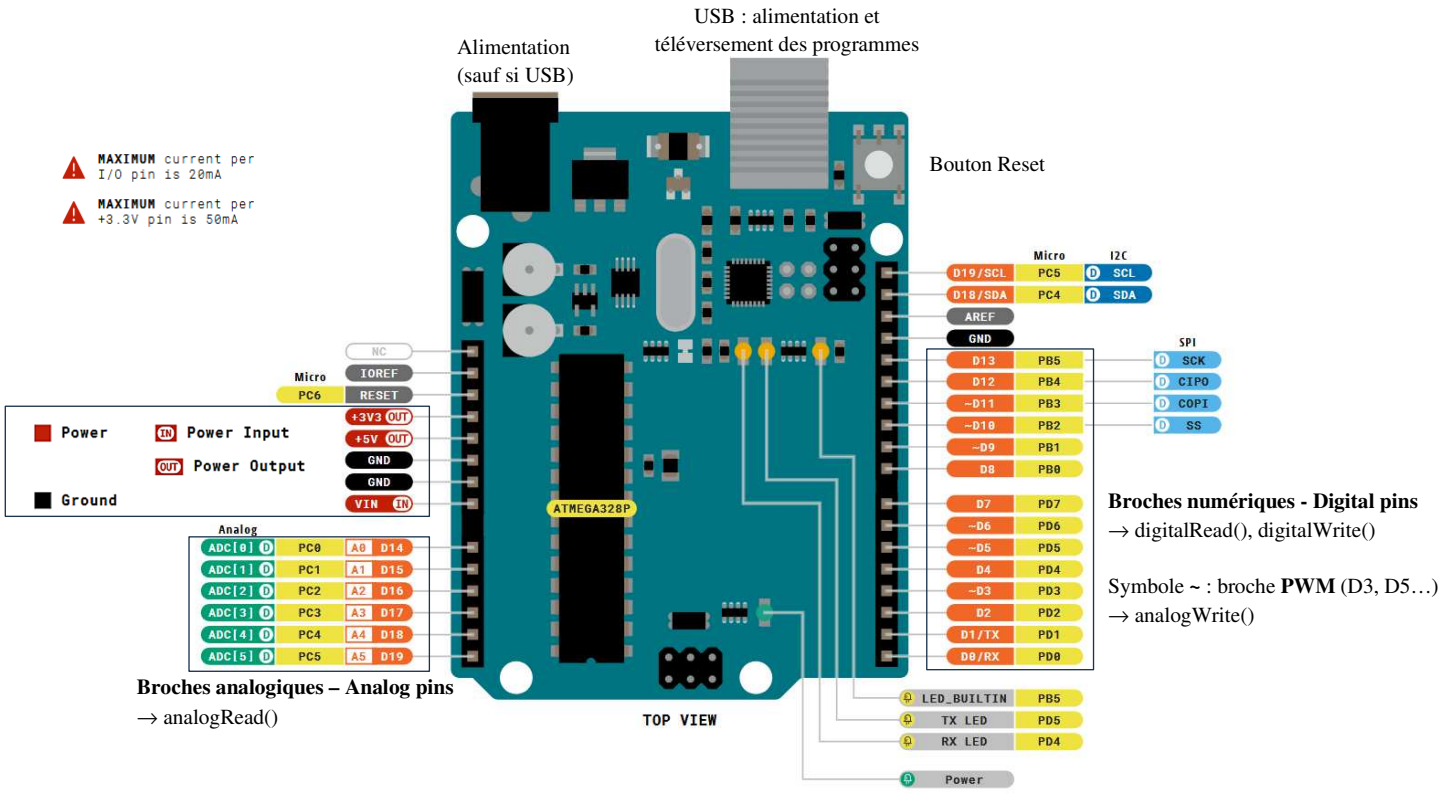
Un microcontrôleur peut être programmé pour analyser et produire des signaux électriques.
Ils sont utilisés dans les ***systèmes embarqués*** pour piloter des robots, dans les voitures, les avions, les récepteurs GPS, les télécommandes, l'électroménager, les jouets, la téléphonie mobile, la domotique,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

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`.

Conversion analogique - Numérique

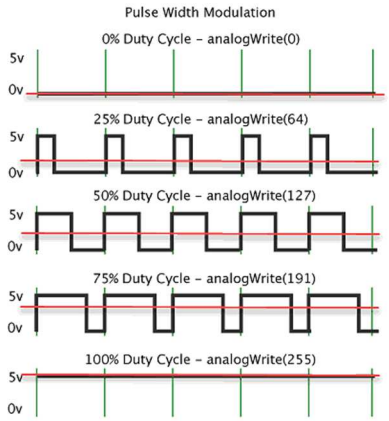
Convertisseur analogique numérique (CAN ou ADC pour analog-to-digital-converter en anglais) *n* bits : une grandeur analogique appliquée au CAN est discrétisée sur 2^n valeurs (de 0 à 2^n -1).

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 faudra donc ***convertir*** la valeur binaire en fonction des caractéristiques du signal envoyé pour accéder à la valeur physique de la grandeur mesurée.

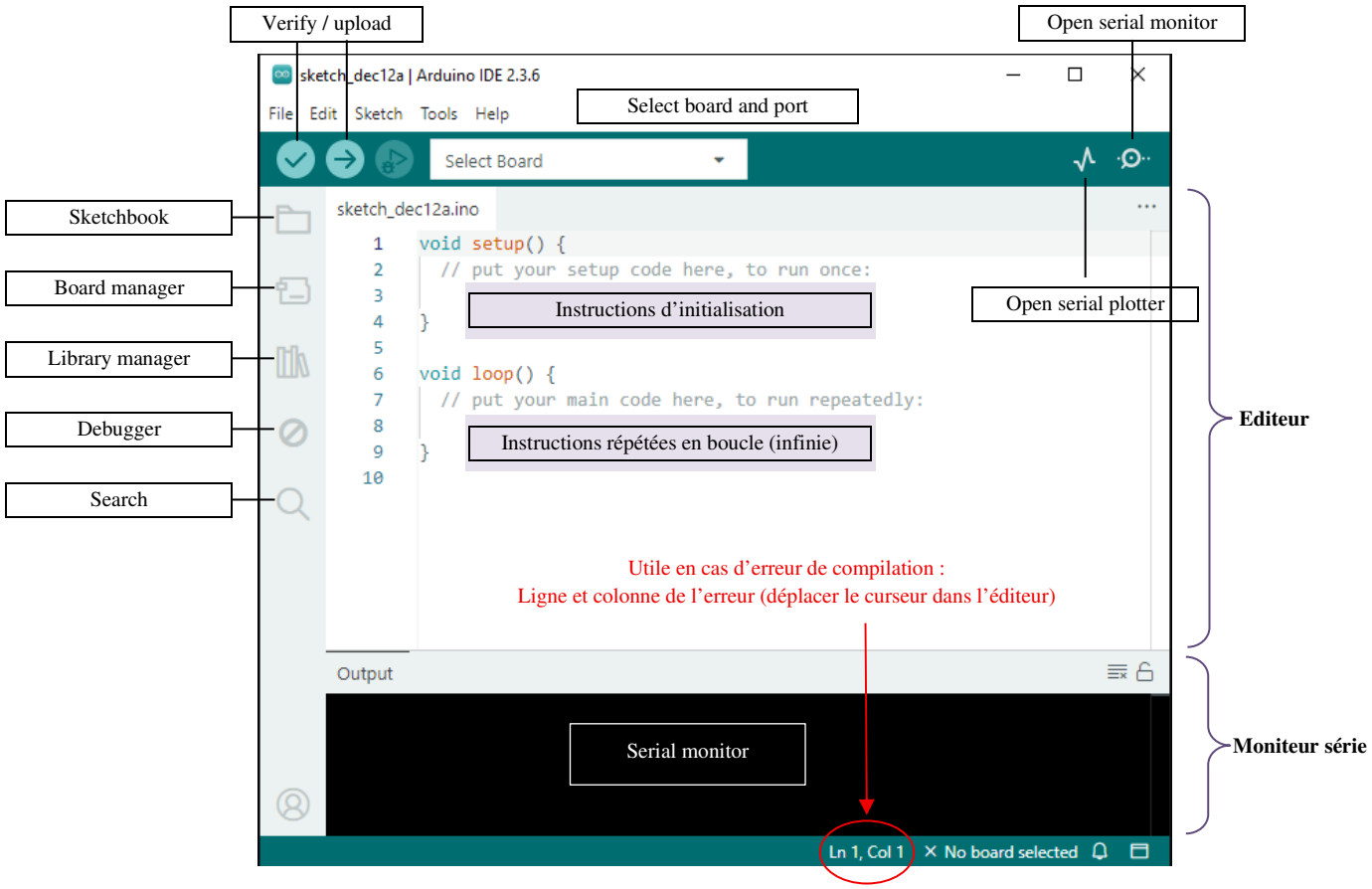
Modulation de largeur d'impulsion (sorties PWM)

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.



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…).

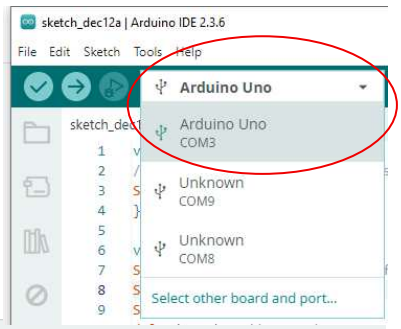


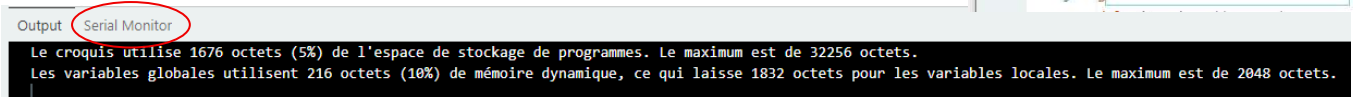
Sketchbook : programmes enregistrés sur l'ordinateur.
Board manager : packages nécessaires avec certaines cartes d'extension ajoutées sur la carte Arduino (WiFi,…).
Library manager : librairies/modules à utiliser pour les capteurs par exemple.
Serial moniter : visualisation du flux de données issu de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

Serial Plotter : visualisation de graphes.

Connexion de la carte à 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.

Mémento langage Arduino C++

```
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. « Découvrir en expérimentant » et tutoriels détaillés.

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 » – « Découvrir en expérimentant » et tutoriels détaillés.

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.