Arduino
Premières manipulations. . .

A. DEMOLLIENS

Chapitre 1

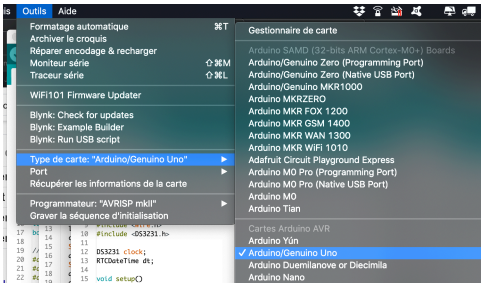
Communiquer...

1 Édition d'un programme

1.1 Premiers réglages

Connectez la carte Arduino à l'ordinateur via le cable USB et démarrez le logiciel Arduino (icône ci-contre). En fait, l'ordre importe peu.
Une fois le logiciel démarré :

- allez dans le menu *Outils* puis *Type de carte* et vérifiez que *Arduino Uno* est bien coché (le cocher sinon) ;
- vérifiez que la carte est bien reconnue en allant dans *Outils*, *Port*. La carte est reconnue si un port est affiché, par exemple "Com 3" ou `/dev/cu.usbmodem14201`.



,

1.2 Choix du débit sur le port série

Pour éviter bien des déboires lors de l'utilisation dans la "vraie vie" au lycée, il me semble important de définir, une bonne fois pour toute la vitesse de communication.
Les choix les plus "classiques" sont :

- 9600 baud... pour les nostalgiques du minitel, et valeur par défaut de la plupart des programmes Arduino ;
- 115200 baud... plutôt à la mode maintenant.

Restons, ici, sur 9600 baud.

1.3 Structure d'un programme

A l'ouverture du logiciel, on peut remarquer qu'un bout de code est déjà proposé :

Listing 1.1 – Programme vide

```
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
```

On dispose, ici, de la structure de tout programme Arduino :

- une fonction (ou procédure) **setup** d'initialisation qui ne sera exécutée qu'une seule fois ;
- une fonction (ou procédure) **loop** qui, comme son nom l'indique également, sera exécutée dans une boucle infinie.

2 Communiquer

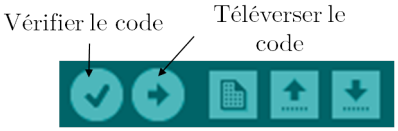
2.1 Le fameux "Hello Word"

On désire afficher, sur l'écran de l'ordinateur, une seule fois "Hello Word".
Modifiez le programme afin d'obtenir le code suivant (sans oublier les ; et en respectant la casse).

Listing 1.2 – Hello world

```
1 void setup() {
2   Serial.begin(9600) ; // vitesse de communication sur le port
       série
3   Serial.println("Coucou !") ;
4 }
5
6 void loop() {
7 }
```

Une fois le code créé, cliquez sur l'icône de vérification du code (corrigez le code si nécessaire !).



Téléverser ensuite le programme dans la carte, puis sélectionnez *Outils/Moniteur série* ou cliquez sur l'icône loupe à droite de la barre d'icônes. Si tout se passe bien, le message apparaît (assurez vous qu'une vitesse de communication à 9600 bauds est bien sélectionnée dans le moniteur série.)

Vous pouvez tenter une variante en plaçant le code de la ligne 3 dans la boucle **loop** !

2.2 Un échange d'information bidirectionnel

Le moniteur série de l'Arduino permet non seulement d'afficher des messages reçus de l'arduino mais également d'en envoyer.

Listing 1.3 – Echo

```
1  void setup() {
2    Serial.begin(9600) ;
3    Serial.println("Je suis prêt !") ;
4  }
5
6  void loop() {
7    if (Serial.available())
8    {
9      String message = Serial.readString() ;
10     Serial.println(message) ;
11   }
12 }
```

Dans la zone de texte en haut de l'éditeur série introduisez votre texte et cliquez sur *Envoyer* ; le message apparait alors dans la zone de réponse.

2.3 Un modèle de programme pour la suite...

Ouvrir le programme `Modele_v1` ;compiler et télécharger sur la carte.

Ce programme peut nous servir, par la suite, pour communiquer avec le moniteur série, pour afficher des résultats par exemple mais surtout pour interpréter les ordres donnés depuis le moniteur série.
Les ordres que l'on sera amenés à envoyer à la carte peuvent être :

Ordre envoyé	Signification
G	GO : on exécute en continu
S	STOP : on arrête
O	ONE : on exécute une seule fois le travail souhaité
I	INFO : on récupère une information sur le programme
Dxxx	DELAI : on fixe un délai entre 2 actions de xxx milli-secondes

Listing 1.4 – Modèle_v1

```
1  // importation des bibliothèques
2
3  // variables utilisées par tous les programmes
4  int BAUD = 9600 ;
5  String info = "Programme modèle communication" ;
6  long delai = 1000 ; // en ms
7
8  void setup()
9  {
10   Serial.begin(BAUD) ;
11   Serial.println(info) ;
12 }
13
14 void loop()
15 {
16   if (Serial.available())
17   {
18     String message = Serial.readString() ;
19     parse(message) ;
20   }
21 }
22
23 /********* INTERPRETATION DES MESSAGES RECUS ************/
24
25 void parse(String msg)
26 {
27   msg.replace("\r", "") ;
28   msg.replace("\n", "") ;
29   char ordre = msg[0];
30   msg.replace(String(ordre), "") ;
31   int valeur = 0 ;
32   if (msg.length() > 0)
```

```
33    {
34      valeur = msg.toInt() ;
35    }
36    switch (ordre) // évite une succession de if, else..
37    {
38      case 'I' :
39        Serial.println(info) ;
40        break ;
41      case 'G' : // exécution d'une tâche régulièrement
42        go() ;
43        break ;
44      case 'O' : // exécution d'une tâche unique
45        job() ;
46        break ;
47      case 'S' : // on stoppe
48        stop() ;
49        break ;
50      case 'D' : // réglage du délai
51        delai = valeur ;
52        break ;
53    }
54  }
55
56  /********** GO - STOP *************/
57
58  void go()
59  {
60  }
61
62  void stop()
63  {
64  }
65
66  /********** JOB *************/
67
68  void job()
69  {
70    Serial.println("Je bosse !") ;
71  }
```

Si on tape `I`, ou `O` dans le moniteur série ; l'information souhaitée apparait dans la console du moniteur série.

Enregistrer le programme précédent sous un autre nom : `Modele_v2` par exemple.

Exo 1 Stop & Go Modifier le programme précédent afin que la fonction `job()` soit appelée régulièrement (sans aucun délai entre les appels pour l'instant) lorsque l'on tape `G` et que le programme "s'arrête" en tapant `S` (en fait, le programme ne s'arrête jamais, c'est la fonction `job()` qui n'est plus appelée).

Passons au chapitre suivant pour fixer un délai entre ces appels.

Chapitre 2

Je suis le maître du temps...

1 Les instructions millis() et delay()

1.1 millis()

Cette instruction permet de connaitre le temps écoulé (en milli-secondes) depuis le moment où la carte est sous tension.
L'instruction `unsigned long time = millis() ;` permet de stocker dans la variable *time* ce temps.

Exo 1 millis() Inclure cette instruction dans un programme élémentaire afin d'afficher le temps à chaque appel de la boucle `loop()`.

1.2 delay()

L'instruction `delay(xxx)` permet d'imposer au micro-controleur une pause de xxx milli-secondes avant de passer à la ligne suivante. C'est une instruction bloquante pour le système.

Exo 2 delay() Ajouter dans le programme précédent une instruction `delay` afin de faire une pause de 1 seconde dans la boucle `loop`.

2 Une fonction job appelée à intervalle de temps régulier

2.1 Un petit délai !

Exo 3 Modèle avec délai Modifier, à l'aide d'une instruction `delay` le programme `Modele_v2` du chapitre précédent afin d'avoir un appel de la fonction `job` tous les *delai* ms. Ajouter un affichage du temps dans la fonction `job` et tester votre programme.

1000 ms passe assez vite et l'on a quand même moyen de "reprendre" la main sur l'exécution du programme dans la boucle `loop` ; mais imaginons une pause de 1 minutes voir plus ; le système resterait ainsi bloqué entre deux appels de la fonction `job` sans que l'on puisse faire quoi que ce soit... il y a mieux à proposer !

2.2 Comptons le temps qui passe

L'idée est, dans la boucle `loop` de compter le temps passé depuis le dernier appel de la fonction `job` ; si l'intervalle de temps est supérieur à *delai* on appel la fonction ; sinon on ne fait rien.

Exo 4 Modèle travail Modifier à nouveau le programme précédent en implémentant un décompte du temps passé.

Bravo, nous obtenons, ici, un programme tout à fait intéressant pour la suite !

2.3 Pour les pros... utiliser une interruption

En fait, il y a au cœur du microcontrôleur un circuit (plusieurs en réalité) qui permet de gérer des tâches à intervalle régulier : il s'agit du **Timer**. Celui-ci, à intervalle de temps régulier va bloquer le système et lancer l'exécution d'une fonction que l'on précisera.

Listing 2.1 – Programme Timer

```
1  #include<MsTimer2.h>
2
3  long delai = 1000 ;
4
5  void setup()
6  {
7    Serial.begin(9600) ;
8    MsTimer2::set(delai,job) ;
9    MsTimer2::start() ;
10 }
11
12 void loop()
13 {
14   fonctionQuiPrendDuTemps() ;
15 }
16
17 void job()
18 {
19   Serial.println(millis()) ;
20 }
21
22 void fonctionQuiPrendDuTemps()
23 {
24   long randomDelay = random(50,800) ;
25   delay(randomDelay) ;
26 }
```

L'importation d'une bibliothèque (ligne 1) est nécessaire pour l'utilisation du timer. Nous utilisons, ici, MsTimer2. Le code de cette bibliothèque est à placer dans le dossier `libraries` du dossier `Arduino`.
Ligne 8, on initialise ce timer en lui précisant un intervalle de temps (en ms) et le nom de la fonction à exécuter (`job` ici).
Ligne 9 : on déclanche le Timer... le code contenu dans la fonction `job` est alors exécuté tous les *delai* ms. On arrêterait le timer avec l'instruction `MsTimer2::stop()`.
Et... victoire, ça fonctionne plutôt bien ! Sur la console du moniteur série on lit 998, 1998, 2999, 3999, 5000, 6000, 7001...Ce n'est pas encore une horloge atomique, on se décale environ d'une ms toutes les s, mais c'est déjà pas si mal !

L'utilisation d'un timer est certes séduisante mais ne peut, malheureusement, pas être utilisée systématiquement. En interne, le fonctionnement de ce timer nécessite l'utilisation de certaines ressources machines. Il est donc impossible d'utiliser ces mêmes ressources dans la routine appelée par le timer !

Ainsi :

- on ne peut pas utiliser d'instruction `delay(...)` avec un timer ;
- certaines sorties PWM ne sont pas compatibles ;
- certains capteurs sur port I2C peuvent ne pas fonctionner.

Exo 5 Modèle_Timer Modifier le programme `Modele` afin que l'appel de la fonction `job` soit géré, cette fois par un timer.

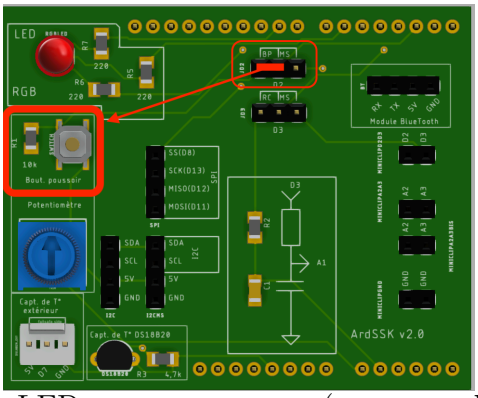
2.4 Maître du temps... pas tout à fait quand même !

Les solutions proposées précédemment nous permettent de gérer relativement facilement le temps sur une échelle de quelques millisecondes à, disons, une heure. Mais...

- en dessous de la milliseconde, nous risquons vite d'atteindre les limites, en terme de rapidité, de la carte `Arduino Uno` et des capteurs dont nous disposons facilement. Des solutions pour travailler à l'échelle de la microseconde existent mais, il faut y mettre le prix !
- au-delà de plusieurs heures d'utilisation, la solution proposée n'est plus très efficace. Si on veut, par exemple, suivre l'évolution de la température sur une journée avec une mesure toute les heures, nous risquons une dérive du temps et, surtout, la moindre micro coupure dans l'alimentation conduira à une réinitialisation du temps. Une solution « simple » consiste à associer à la carte Arduino une horloge en temps réel.

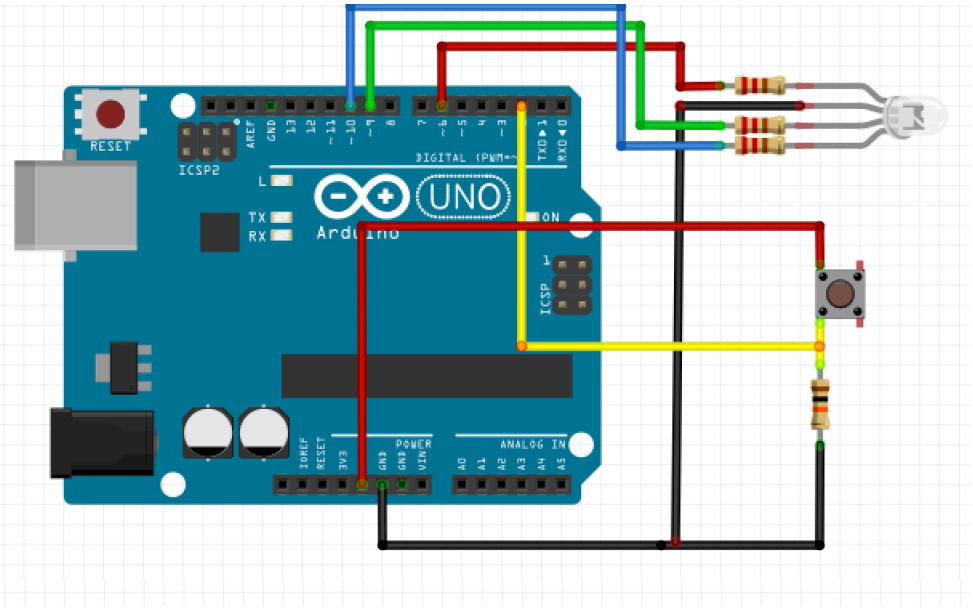
Chapitre 3

Sorties numériques et analogiques



Pour les utilisateurs de la carte d'apprentissage, s'assurer que le cavalier D2 est placé du côté du bouton poussoir.
La LED RGB est connectée aux ports D6 (rouge), D9 (bleu) et D10 (vert).
L'état du bouton poussoir sera lu sur D2.
La LED RGB proposée est une LED à cathode commune (reliée à la masse). Appliquer un potentiel positif sur l'une des pattes de la LED permet d'allumer la couleur correspondante.

Pour les autres, réaliser le montage ci-dessous... avec une seule LED pour commencer (par exemple connectée sur le port D6).



1 Sortie numérique

1.1 L'instruction digitalWrite

Listing 3.1 – Programme arnaque

```
1  int RED = 6 ;
2
3  void setup()
4  {
5    pinMode(RED, OUTPUT) ;
6    digitalWrite(RED, HIGH) ;
7  }
8
9  void loop()
10 {
11 }
```

Exécuter ce programme.
Et bien, c'est magnifique... ce programme ne fait rien, mais le fait bien !
En fait, la situation aurait pu être pire, vous auriez pu, par exemple oublier un point virgule à la fin d'une ligne, ou écrire red sans respecter la casse, ou autre...

Remplacez maintenant `LOW` par `HIGH` dans le programme précédent et *fiat lux* !

L'instruction `pinMode(numero_port, OUTPUT) ;` permet de spécifier que le port en question est une sortie. Les ports analotiques D0 à D13 peuvent être utiliser soit comme entrée numérique, soit comme sortie numérique ; il est bon de spécifier l'utilisation.

1.2 LED Blinking !

Exo 1 Clignotement Écrire un programme permettant de faire clignoter la LED rouge.

- vous avez, dans un premier temps droit à deux instruction `digitalWrite` ;
- vous n'avez droit qu'à une seule instruction `digitalWrite` (on précise que les valeurs `HIGH` et `LOW` peuvent être stockés dans une variable de type `int`.

Exo 2 Feu tricolore Si vous disposez de trois LED ou d'une LED RGB, faire apparaitre successivement et périodiquement les 3 couleurs.

1.3 Commande de la LED, version software

Exo 3 Allumer/éteindre Modifier un des programmes modèles afin d'allumer la LED quand on tape `G` et l'éteindre quand on tape `S`.

2 Sortie analogique

2.1 L'instruction analogWrite

Changeant la ligne 6 du programme 3.1 du paragraphe précédent par `analogWrite(RED, 80);`

On obtient, en principe, un éclairement moindre de la LED.
Charger le programme `LED_Variable` dont le code est donné ci-dessous.

Listing 3.2 – Programme LED_Variable

```
1  int RED = 6 ;
2
3  void setup()
4  {
5    pinMode(RED, OUTPUT) ;
6  }
7
8  void loop()
9  {
10   for (int i = 0 ; i < 255 ; i = i + 10)
11   {
12     analogWrite(RED, i) ;
13     delay(200) ;
14   }
15 }
```

Au lieu d'appliquer une tension soit nulle soit égale à 5 V sur la LED, on impose, grâce à la fonction **analogWrite** une tension « moyenne » égale à $\frac{i \times 5}{255} V$.

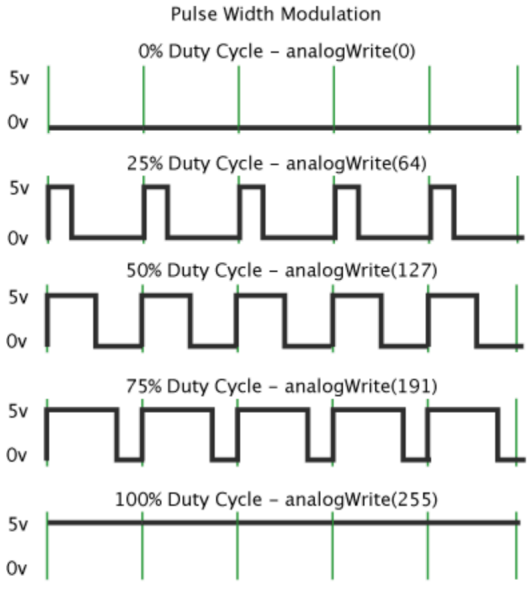
Exo 4 Feu tricolore avec du orange Modifier le programme du feu tricolore précédent afin d'afficher une couleur se rapprochant du orange (par exemple R = 255, G = 165, B = 0).

2.2 Sortie PWM

Un certain nombre de sorties de l'Arduino peuvent être commandés en PWM ; elles sont identifiées par un tilde.

En réalité, ce n'est pas vraiment une tension continue qui est appliquée sur le port correspondant. On a une fonction dite PWM (Pulse With Modulation). Si on demande une tension de 5V, on a, en sortie, 5 V en continu. Mais, si, par exemple on demande une tension de 1 V, le système délivre 5 V sur un cinquième du temps.

Le schéma ci-contre est extrait de la documentation d'Arduino.
L'intervalle de temps entre deux traits verts correspond à 2 ms, la fréquence du PWM étant de 500 Hz.



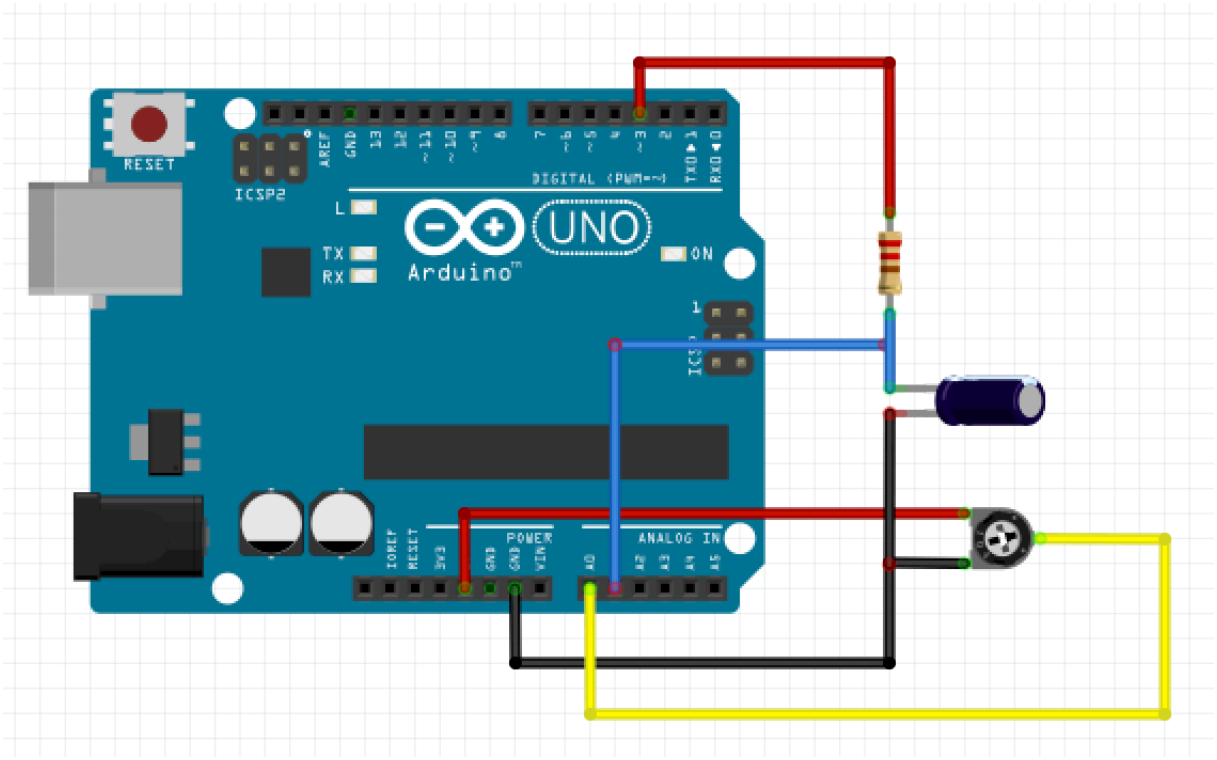
Chapitre 4

Entrées numériques et analogiques

Câbler pour cette partie (pour les non utilisateurs du module d'apprentissage) :

- le bouton poussoir (cf chapitre précédent) ;
- un diviseur de tension (potentiomètre rotatif de 10 kΩ) alimenté entre 0 et 5 V et dont le point milieu est connecté à l'entrée analogique **A0** ;
- un circuit RC alimenté par le port **D3** et dont le point milieu est connecté à l'entrée analogique **A1** R = 10 kΩ et C = 10 μF.

Le schéma du circuit, utile dans ce chapitre est donné ci-dessous.



1 Entrée numérique

1.1 L'instruction digitalRead()

L'instruction `digitalRead(numero_port)` va permettre de tester si la tension sur le port en question est à l'état haut (tension supérieure à 3 V) ou bas. L'état en question peut être stocké dans une variable de type entier ou booléen (1, 0 étant interprété comme True/False).
Lancer le programme suivant et appuyer par moment sur le bouton poussoir. Vous devriez voir s'afficher soit des 0 soit des 1 dans le moniteur série.

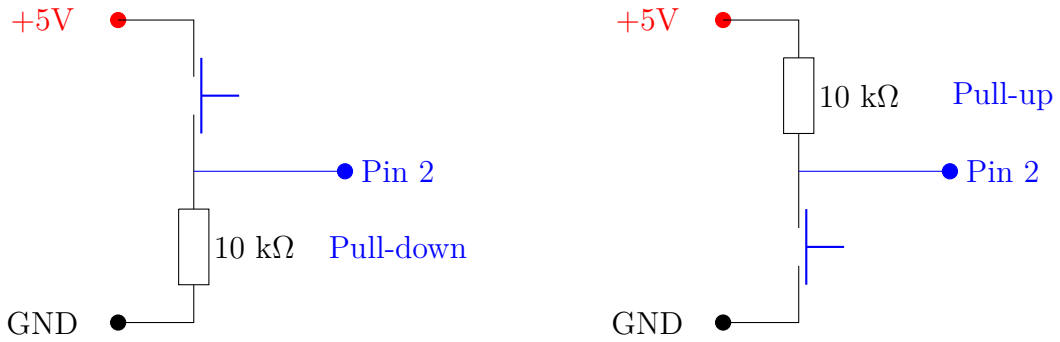
Listing 4.1 – Programme DigitalRead

```
1  int BUTTON = 2 ;
2
3  void setup()
4  {
5    Serial.begin(9600) ;
6    pinMode(BUTTON, INPUT) ;
7  }
8
9  void loop()
10 {
11   int etat = digitalRead(BUTTON) ;
12   Serial.println(etat) ;
13   delay(100) ;
14 }
```

Nous devions spécifier, cette fois, que le port 2 est une entrée.
A chaque parcours de la boucle, on lit l'état du bouton poussoir et on affiche cet état.

1.2 Pull-up ou Pull-down ?

Le câblage du bouton poussoir mérite un peu d'attention au risque de réaliser un court-circuit entre la sortie 5V et la masse... sortie qui ne peut rappelons-le supporter que quelques mA.



Le câblage de gauche est celui que nous avons implanté (montage dit pull-down) :

- lorsque le bouton est relâché, le port 2 est relié à la masse via la résistance de 10 k Ω ;
- lorsqu'il est appuyé, il se retrouve lié au +5 V

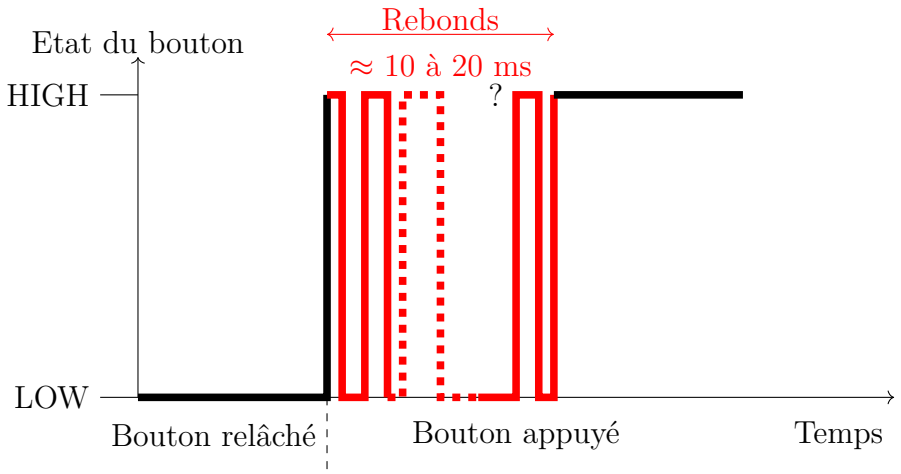
La logique est « inversée » dans le montage de droite (montage dit pull-up) ou, cette fois, le port est à l'état + 5V lorsque le bouton n'est pas appuyé.

1.3 Un bouton poussoir en mode continu

Exo 1 Commande de la LED, mode bouton à sonnette Modifier le programme `DigitalRead` afin d'allumer la LED quand le bouton poussoir est appuyé et de l'éteindre sinon.

1.4 Bouton poussoir en mode marche/arrêt

L'utilisation du bouton poussoir comme bascule marche/arrêt n'est pas si simple que cela. C'est un problème bien connu des électroniciens, lorsque l'on appuie sur le bouton poussoir, le contact n'est pas immédiat. Des « rebonds » se produisent durant une vingtaine de millisecondes et... l'état logique du bouton est alors indéfini.



Une solution matérielle est bien connue, il suffit de brancher un condensateur en parrallèle du bouton poussoir. Ainsi, toutes les rapides fluctuations de tension dues aux rebonds seront filtrées.

Quelle solution logicielle proposer ? Plusieurs solutions sont possibles mais, en gros, il faudrait « oublier » tout ce qui se passe pendant les rebonds. Par exemple, on peut repérer le premier front montant, on attend 20 ms et on vérifie que l'état du bouton poussoir est toujours haut. Dans ce cas, on change l'état de la LED.

La boucle du programme pourrait alors correspondre au code suivant.

Listing 4.2 – Programme LED_BoutonMarcheArret avec anti rebond

```
1  void loop()
2  {
3    boolean button = digitalRead(BUTTON) ;
4    if (button != lastButton)
5    {
6      delay(20) ;
7      button = digitalRead(BUTTON) ;
8      if (button == HIGH)
9      {
10       ledState = ! ledState ;
11       digitalWrite(RED, ledState) ;
12     }
13   }
14   lastButton = button ;
15 }
```

En fait... nous nous compliquons la vie pour rien car, la plupart du temps, la carte Arduino sera connecté à l'ordinateur et la solution, de loin la plus simple est d'utiliser le programme réalisé à la fin du chapitre précédent avec une commande de la LED à partir du moniteur série.

2 Entrée analogique

La platine Arduino Uno dont vous disposez comporte 6 ports dédiés à une lecture analogique (nommés A0, A1... () A5). Il s'agit de convertisseurs analogique-numérique (ADC) sur 10 bits (2^{10} = 1024). On peut lire une tension, en entrée, comprise entre 0 et 5 V, ce qui donne une résolution de 5 mV. La conversion demande 100 microsecondes.

2.1 L'instruction analogRead()

L'instruction `analogRead(numero_port)` permet de récupérer, sous forme d'entier, une valeur comprise entre 0 et 1023.

Charger le programme suivant et tourner le bouton du potentiomètre. Le résultat de la conversion analogique numérique (sur 10 bits) est alors affiché.

Listing 4.3 – Programme AnalogRead

```
1  int POTAR = A0 ;
2
3  void setup()
4  {
5    Serial.begin(9600) ;
6  }
7
8  void loop()
9  {
10   int value = analogRead(POTAR) ;
11   Serial.println(value) ;
12   delay(500) ;
13 }
```

2.2 Un potentiomètre pour faire varier l'intensité

Exo 2 Eclairage variable Modifier le programme précédent afin de faire varier l'intensité lumineuse de la LED lorsqu'on tourne le bouton du potentiomètre. Atttention l'instruction `analogRead` retourne une valeur comprise entre 0 et 1023 alors que `analogWrite` attend une valeur comprise entre 0 et 255 !

2.3 Programme d'acquisition

Exo 3 Premier programme d'acquisition Modifier un des programmes modèles afin d'obéir au cahier des charges suivant :

- lorsque l'on envoie l'instruction `G` on lance l'acquisition ; l'instant correspondant devra correspondre, par la suite au temps 0 ;
- lorsque l'on envoie l'instruction `S` on stoppe l'acquisition ;
- l'acquisition correspond à la lecture du point milieu du potentiomètre ;
- l'affichage se fera sous la forme `TIME:ttt:A0:xxx`.

Pour obtenir un tel affichage, on pourrait se baser sur les lignes de code suivantes :

```
1 String msg = "TIME:" + String(millis()) ;
2 msg = msg + ":A0:" + String(analogRead(POTAR)) ;
3 Serial.println(msg)
```

2.4 Exploitation de la sortie formatée

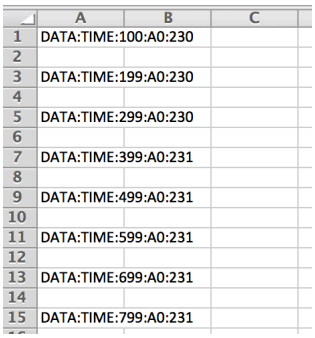
Le programme précédent pourra très bien servir de modèle pour vos futurs TP car... tout y est :

- la gestion du temps ;

- la gestion de l'acquisition ;
- une sortie formatée.

a Copier dans Excel

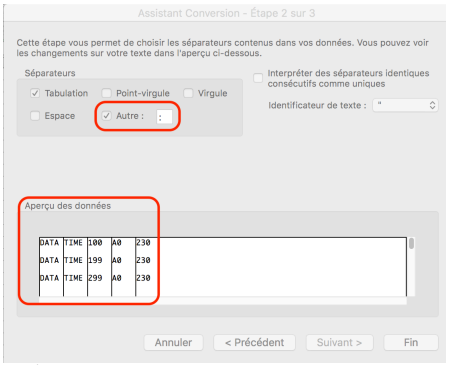
Faire une acquisition sur quelques secondes en tournant le bouton du potentiomètre rotatif.



	A	B	C
1	DATA:TIME:100:A0:230		
2			
3	DATA:TIME:199:A0:230		
4			
5	DATA:TIME:299:A0:230		
6			
7	DATA:TIME:399:A0:231		
8			
9	DATA:TIME:499:A0:231		
10			
11	DATA:TIME:599:A0:231		
12			
13	DATA:TIME:699:A0:231		
14			
15	DATA:TIME:799:A0:231		

Faire un copier coller de l'intégralité des données (ou de celles qui nous intéressent) du moniteur série d'Arduino vers la première case d'une feuille Excel.

Les données sont maintenant dans la première colonne, éventuellement séparées d'une ligne blanche.



Sélectionnez la première colonne puis, dans le menu **Données** d'Excel, cliquez sur l'item **Convertir...**.

Une boite de dialogue s'ouvre, passez à la seconde page et préciser que le séparateur est **:** (ou autre si vous avez choisi un autre format de sortie).

Les données apparaissent alors dans chacune des premières colonnes.

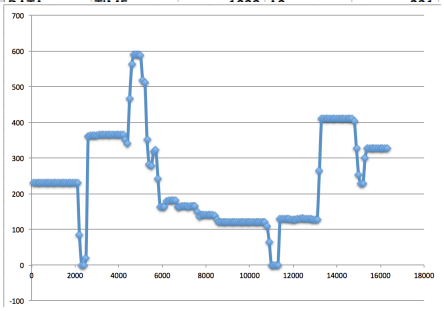
	A	B	C	D	E
1	DATA	TIME	100	A0	230
2					
3	DATA	TIME	199	A0	230
4					
5	DATA	TIME	299	A0	230
6					
7	DATA	TIME	399	A0	231
8					
9	DATA	TIME	499	A0	231
10					

Sélectionnez l'ensemble des données et cliquez sur le bouton **Trier** .

	A	B	C	D	E
1	DATA	TIME	100	A0	230
2	DATA	TIME	199	A0	230
3	DATA	TIME	299	A0	230
4	DATA	TIME	399	A0	231
5	DATA	TIME	499	A0	231
6	DATA	TIME	599	A0	231
7	DATA	TIME	699	A0	231
8	DATA	TIME	799	A0	231
9	DATA	TIME	899	A0	231
10	DATA	TIME	999	A0	231

Nous avons ainsi nos données prêtes à être traitées.

Si l'on a des nombres réels, il faudra penser à convertir les . en , ! .



Reste alors à faire la représentation graphique...

b Exploiter les données en python

On peut très bien également copier les données du moniteur série et les sauvegarder dans un fichier texte (data.txt par exemple) .

Le programme acquisition.py permet de lire les données dans un fichier texte que l'on vient de créer, de les interpréter et de tracer la courbe.

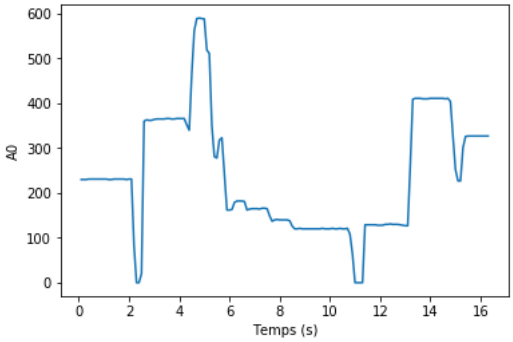
Listing 4.4 – Programme acquisition.py

```
1  import matplotlib.pyplot as plt
2
3  fichier = open('data.txt')
4  lignes = fichier.readlines()
5  fichier.close()
6
7  Temps=[] # temps en s
8  A0=[]    # analogRead sur port A0
9
10 for ligne in lignes :
11   ls = ligne.split(':') # on découpe la ligne
12   if len(ls) >= 4:
13     Temps.append(float(ls[1])/1000)
14     A0.append(int(ls[3]))
15
16 plt.plot(Temps, A0)
17 plt.xlabel('Temps (s)')
18 plt.ylabel('A0')
19 plt.show()
```

Ce programme n'est absolument pas générique mais permet d'être facilement adaptable :

- On ouvre le fichier data.txt et on extrait dans la variables *lignes* l'ensemble des lignes du fichier. Cette variable est une liste de chaines de caractères. Le premier élément de la liste correspond à 'TIME :100 :A0 :230'.
- On crée 2 listes vides pour stocker les données.
- L'instruction clé est la ligne 11 qui permet de couper chacune des lignes à l'occurence du caractère **:**. On crée ainsi une variable *ls* qui est à nouveau une liste de chaines de caractère. Pour la première ligne *ls* correspond à ['TIME','100','A0','230'].
- l'élement d'indice 1 correspond à la valeur du temps, celui d'indice 3 à la lecture sur A0 ; reste à convertir la chaine de caractère en réel ou entier et stocker les valeurs.

On dispose alors de deux tableaux de valeurs que l'on peut exploiter en python.



Et si on voulait tracer la tension plutôt que la lecture du port analogique, il vaudrait mieux,dans la mesure du possible, faire exécuter les calculs par l'ordinateur. L'Arduino ne dispose pas d'unité arithmétique, de ce fait, les calculs réalisés sur le microcontrôleur demande beaucoup de temps.

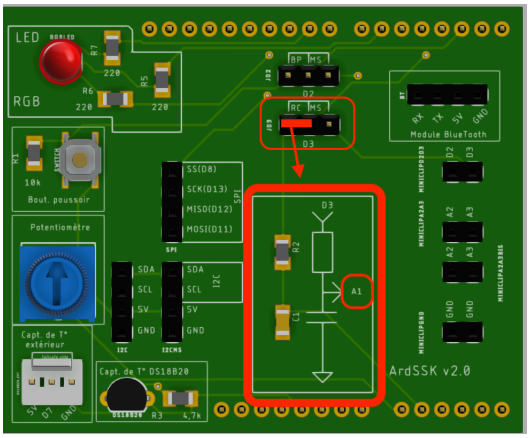
Chapitre 5

Et en TP dans la vraie vie ?

1 Un premier TP : décharge d'un circuit RC

Nous allons utiliser, ici, le circuit électrique câblé ou sur la platine (un circuit RC alimenté par le port **D3** et dont le point milieu est connecté à l'entrée analogique **A1** comme reporté sur le schéma en tête du chapitre précédent).

Pour cette partie, il faut mettre le cavalier `D3` de la platine du kit vers la gauche, comme sur la figure ci-contre.



1.1 Que faut-il faire ?

Nous exploitons ici l'entrée analogique **A1** et la sortie analogique/numérique **D3**. A un facteur multiplicatif près $\left(\frac{5000}{1023}\right)$, **A1** permet de lire la ddp (en mV) aux bornes du condensateur.
Supposons que l'on souhaite étudier la décharge du condensateur. Que souhaite-t-on faire ?

1. Imposer sur l'entrée **3** une tension de 5V et attendre (par exemple 2 ou 3 s) que le condensateur soit chargé.
2. A l'instant t=0, on applique une tension nulle sur l'entrée **3**.
3. On mesure la tension sur l'entrée **A1** et on envoie les informations sur le port série.
4. On arrête au bout d'un certain temps !

Côté Arduino, on sait programmer chacune de ces tâches individuellement :

1. digitalWrite(HIGH) et delay(3000) feront l'affaire ;
2. digitalWrite(LOW) et start = millis() ;
3. valeur = analogRead(A1) et println(TIME :ttt :A0 :xxx) bien sûr ;
4. on peut, par exemple, s'arrêter quand l'entrée analogique devient inférieure à 10 (la tension atteint alors environ le centième de sa valeur initiale).

1.2 Une solution « simple »...

Le programe `RC_Minimum` propose une implémentation des 4 étapes précédentes.

Listing 5.1 – Programme RC_Minimum

```
1  int READ = A1 ;
2  int COMMANDE = 3 ;
3  long start = 0 ;
4  int sensorValue = 0 ;
5
6  void setup()
7  {
8    Serial.begin(9600);
9    pinMode(COMMANDE, OUTPUT) ;
10   digitalWrite(COMMANDE, HIGH) ;
11   delay(3000) ;
12   sensorValue = analogRead(READ);
13   digitalWrite(COMMANDE, LOW) ;
14   start = millis() ;
15 }
16
17 void loop()
18 {
19   if (sensorValue > 10)
20   {
21     sensorValue = analogRead(READ);
22     String msg = "TIME:" + String(millis() - start) ;
23     msg = msg + ":A0:" + String(sensorValue) ;
24     Serial.println(msg) ;
25   }
26 }
```

On va :

- Faire toutes les initialisations dans la fonction **setup**, y compris la charge du condensateur. Un premier appel d'`analogRead` permet d'initialiser la variable $sensorValue$ à une valeur proche de 1023.
- Lignes 13 et 14, on impose 0 V sur le port **3** et on note le temps.
- Dans la boucle, si la valeur lue est supérieure à 10, on lit à nouveau la valeur et on envoie les résultats formatés sur le port série.

Et ensuit... on exploite les résultats comme à la fin du chapitre précédent.

1.3 Commande à partir du moniteur série

Exo 1 RC version 2 Modifier le programme d'acquisition du chapitre précédent de telle sorte que :

- on charge le condensateur après avoir envoyé l'instruction `C` ;
- on étudie la décharge lorsque l'on envoie l'ordre `O`.

1.4 Et pour aller plus loin !

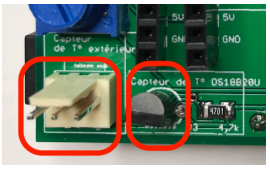
Exo 2 Temps de demi décharge

Modifier la fonction `job` du programme précédent pour afficher, lors d'une décharge, uniquement le temps au bout duquel la tension aux bornes du condensateur atteint la moitié de la valeur initiale.

EXO 3 Étude en fonction de la tension de charge
La broche **3** qui permet d'alimenter le circuit RC que l'on étudie est une sortie PWM. Modifier le programme de l'exercice précédent afin d'afficher dans le moniteur série tension et temps de demi décharge pour une dizaine de valeurs de la tension. Pour chaque valeur de la tension, on pourra éventuellement faire la moyenne de 10 mesures du temps de demi décharge.

2 Utilisation d'un capteur et de sa bibliothèque

Nous prenons l'exemple du capteur de température DS18B20 dont un composant est soudé sur la platine d'essai.



2.1 La bibliothèque DallasTemperature

L'utilisation de la bibliothèque DallasTemperature est alors nécessaire.
Il faudra installer cette bibliothèque sur votre ordinateur !
Un exemple d'utilisation est proposé ; il est précieux car il précise comment communiquer avec le capteur. Il faudra alors l'adapter à vos besoins.

2.2 Il nous faut l'adapter à nos besoins

En faisant un peu le ménage dans cette bibliothèque on pourrait concevoir le programme suivant.

Listing 5.2 – Acquisition température

```
1  #include <DallasTemperature.h>
2  #include <OneWire.h>
3
4  int ONE_WIRE_BUS = 7 ; # sur notre platine d'essai
5  OneWire oneWire(ONE_WIRE_BUS);
6  DallasTemperature sensors(&oneWire);
7
8  void setup()
9  {
10   Serial.begin(115200);
11   sensors.begin();
12 }
13
14 void loop()
15 {
16   sensors.requestTemperatures();
17   float T = sensors.getTempCByIndex(0);
18   String msg = "TIME:" + String(millis() ) ;
19   msg = msg + ":T:" + String(T, 1) ;
20   Serial.println(msg) ;
21   delay(1000) ;
22 }
```

3 Et un jour peut-être en manip de cours ou TIPE ?

La démarche proposée précédemment peut, bien sûr, se généraliser. L'idée est, avec un "petit peu" d'expérience de pouvoir extraire le code donné en exemple avec une bibliothèque pour l'adapter à nos besoins. Vous pouvez vous reporter au guide pédagogique que j'ai réalisé pour trouver de nombreux exemples de codes.

3.1 Un capteur, différentes utilisation...

Parmi les nombreuses possibilités, on peut signaler :

- l'utilisation d'un écran LCD pour affichage ;
- la réalisation d'un système autonome alimenté sur batterie (écran LCD + bouton poussoir pour commander le système) avec éventuellement sauvegarde des résultats sur carte SD ;
- la réalisation d'un système connecté : comme précédemment mais l'échange des données se fait à l'aide de son smartphone (vous serez surpris de voir qu'une dizaine de lignes de code suffisent pour adapter un de nos programmes modèles par exemple) ;
- ...

3.2 Différents capteurs...

Les possibilités sont énormes et relativement faciles d'accès.

- les capteurs de température sont de bons candidats pour se "faire la main" ;
- on peut ensuite passer à des milli-wattmètres, des capteurs de champ magnétiques...
- et un jour... un accéléromètre : importation de la bibliothèque, instanciation du capteur, initialisation dans le `setup`, et quelque chose comme `capteur.readAX, ou readAY ou readAZ` dans la fonction `job` et le tour est joué !

3.3 Et un jour, peut-être, changer de microcontroleur...

A titre personnel, j'utilise les cartes ESP32. La programmation peut se faire dans l'environnement de développement d'Arduino comme si c'était un Arduino. Il suffit juste d'ajouter les bibliothèques correspondantes. Même sans vouloir faire des choses très sophistiquées, on peut vite en avoir "besoin" en physique dès que l'on souhaite dépasser le kHz.

Chapitre 6

Eléments de syntaxe

1 Commandes des Entrées/Sorties

1.1 Affection des Entrées/Sorties

Les affectations des entrées/sorties sont à placer dans la procédure *setup*. Pour affecter une entrée sur une broche :
pinMode(8,**INPUT**) //Affectation de la broche 8 (logique) en tant qu'entrée.
pinMode(A0,INPUT) //Affectation de la broche A0 (analogique) en tant qu'entrée.
pinMode(8,**OUTPUT**) //Affectation de la broche 8 (logique) en tant que sortie.
pinMode(9,OUTPUT) //Affectation de la broche 9 (sortie PWM car marquée par le symbole
// sur la carte) en tant que sortie.

1.2 Lire les entrées

Lorsqu'une broche "digitale" ou logique est configurée en entrée, il est possible de récupérer la valeur correspondante en entrée de cette broche via : **digitalRead(numero_de_broche)**

Cette fonction renvoie 1 si la tension en entrée de la broche est supérieur à 3V, et 0 si la tension est inférieure.
Il est possible de lire une tension sur une broche analogique, via un convertisseur analogique/numérique, en appelant la fonction :**analogRead(numero_de_broche)**

La tension lue doit varier entre 0 et 5V. La valeur renvoyée est un entier contenu entre 0 et 1023.

1.3 Imposer une tension en sortie

Sur une broche logique configurée en sortie, on peut imposer soit 0V soit 5V via :
digitalWrite(numero_de_broche,LOW) //impose 0V
digitalWrite(numero_de_broche,**HIGH**)// impose 5V

Certaines broches spéciales configurées en sortie, et suivies du symbole ~, peuvent délivrer une tension **moyenne** variable grâce à la fonction : **analogWrite(numero_de_broche,val)** avec *val* une valeur entière comprise entre 0 et 255.
La sortie $\frac{val}{255} \times 5\,V$ est obtenue par modulation de largeur d'impulsion (PWM en anglais).

2 Quelques instructions propres au langage Arduino

2.1 Communication avec le moniteur série

`Serial.begin(1115200) ;` : à écrire dans la fonction `setup()`), permet d'initialiser la communication série et de spécifier la vitesse de communication .

`Serial.println("Coucou") ;` affiche "Coucou" dans le moniteur série et passe à la ligne.

`Serial.print("Coucou") ;` affiche "Coucou" dans le moniteur série sans passer à la ligne.

L'instruction `Serial.available()` permet de savoir si des données ont été transmises du moniteur série vers l'Arduino. Si le test précédent est vrai, l'instruction `String msg = Serial.readString()` permet de stocker dans la variable *msg* (de type String) les données en question.

2.2 Le temps...

L'instruction `millis()` permet de connaître le temps écoulé depuis l'initialisation du micro contrôleur.

L'instruction `delay(xxx)` permet d'exécuter une pause de xxx millisecondes.

3 Eléments de syntaxe C++

3.1 Les différents types et les précautions qui vont avec

Contrairement à Python, Arduino requiert qu'on lui renseigne un type à chaque variable créée.
Il faut être vigilant sur le type utilisé en essayant de limiter l'occupation en mémoire.
Les types les plus utilisés sont regroupés dans le tableau suivant :

Type de variable	Intervalle	Commentaires
boolean	*True* ou *False*	Occupe 1 octet en mémoire.
String	chaîne de caractère	Occupe 1 octet/caractère.
int	[-32,768 ;32,767]	Représentation partielle de $\mathbb{N}$ Occupe 2 octets en mémoire.
long	[-2 147 483 648 ;2 147 483 647]	Représentation partielle de $\mathbb{N}$ Occupe 4 octets en mémoire.
float	$[-3,4028235.10^{38}\,;3,4028235.10^{38}]$	Représentation partielle de $\mathbb{R}$ Occupe 4 octets en mémoire.

L'instruction `String(xxx)` permet de convertir la variable xxx en une chaîne de caractères ; la concaténation des chaînes de caractères se fait par + : exemple `String str = "TIME:"+String(millis())`

3.2 Point virgule en fin de ligne !

En C/C++ chaque instruction doit se terminer par un ";" !

3.3 Des accolades pour délimiter un bloc de code

En python on utilise des indentations pour identifier les blocs de code (boucle, fonction, condition...), ici le bloc de code sera délimité par une accolade ouvrante et une fermante. La ligne précédent l'accolade ne sera pas terminée par un ;.

a Boucle *for*

```
1 for(int i=0;i<=10; i = i + 1)
2 {
3     //corps de la boucle
4 }
```

b Boucle *while*

```
1 int i=10;
2 while(i>=0)
3 {
4     //corps de la boucle
5     i = i - 1;
6 }
```

c Structure conditionnelle *if*

```
1  int a=0;
2  int b=2;
3  if(a==b)
4  {
5      Serial.println("a=b");
6  }
7  else if(a>b)
8  {
9      Serial.println("a>b");
10 }
11 else
12 {
13     Serial.println("a<b");
14 }
```