Projet 5 - Températures : Mesurer & Exploiter

Comme la vitesse de propagation des ondes sonores dans l'air dépend de la température, nous allons à présent utiliser notre Arduino pour mesurer des températures.

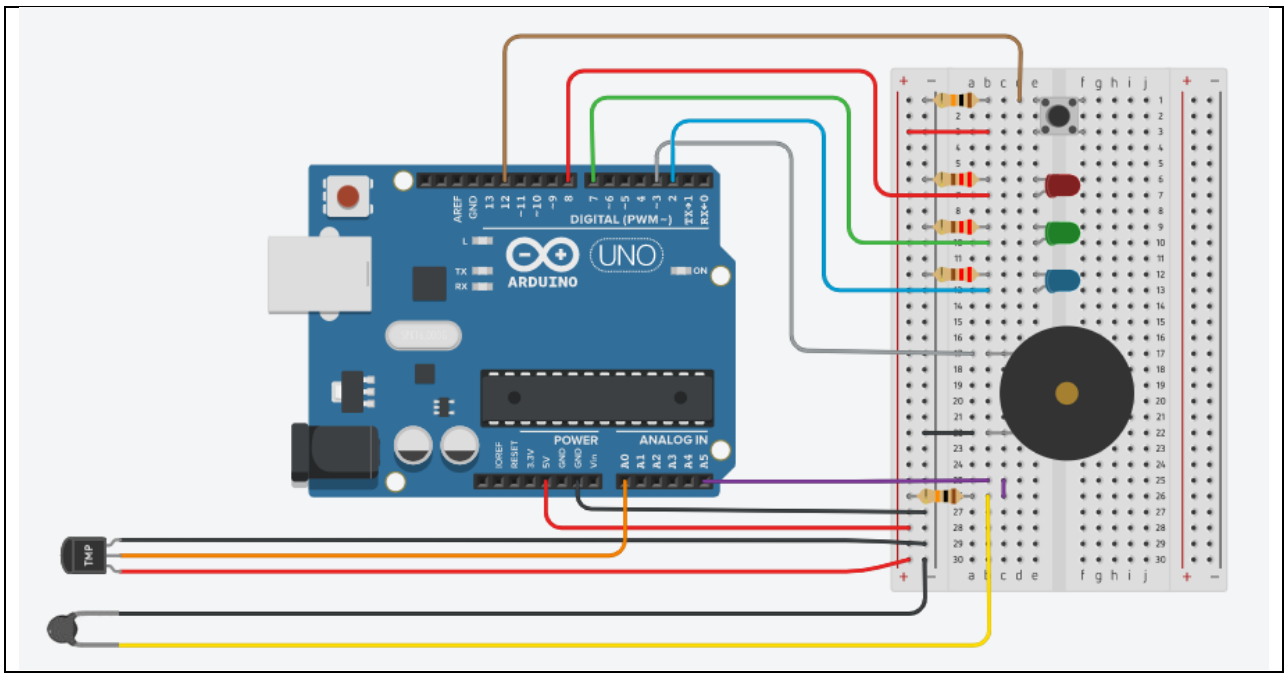
Les capteurs de température généralement utilisés avec un Arduino sont de deux types différents :

- les capteurs dont la tension de sortie est linéairement proportionnelle à la température (les plus courant : TMP 36 et LM 35)

- les thermistances CTN (coefficient de température négatif) qui sont des capteurs de température résistifs, dont la résistance diminue de façon non linéaire quand la température augmente.

Avec un Arduino, on utilise ces capteurs pour réguler la température d'une pièce, surveiller la température de fonctionnement d'un matériel, dans les stations météo…

Toutes les activités de mesure de température et d'exploitation seront réalisées avec le circuit suivant :



- **Liste des composants :**

. 1 capteur de température (TMP 36 ou LM 35)

. 1 thermistance

. 1 DEL rouge

. 1 DEL verte

. 1 DEL bleue

. 3 résistances de 220 Ω (résistance de protection des DELs)

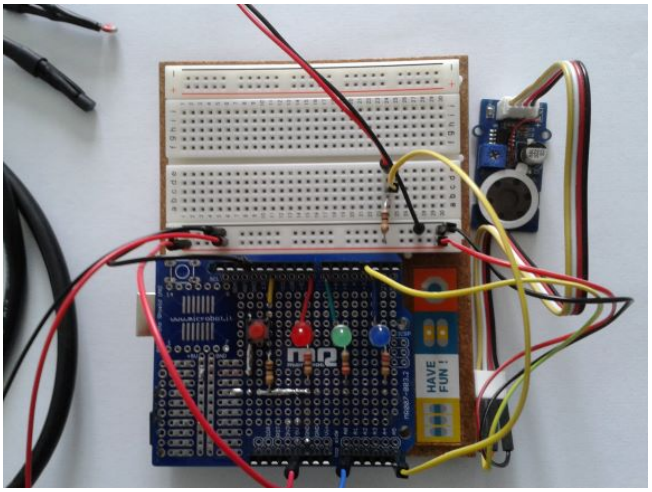
. 2 résistances de 10 kΩ (résistance du bouton poussoir et de la CTN)

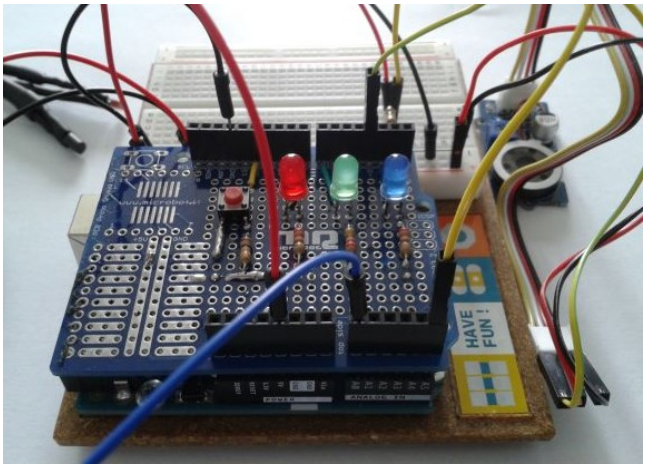
. 1 bouton poussoir

. 1 haut-parleur (ou piezo)

. 1 plaque d'essais

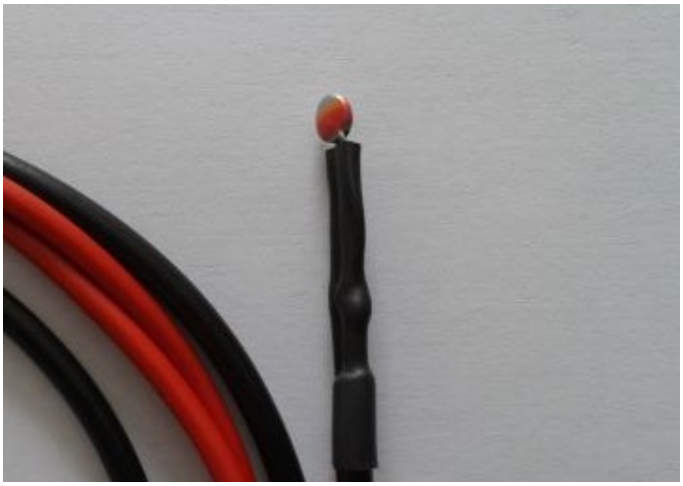
. Fils de connexion





Le circuit sur un ”shield” pour Arduino Uno

Capteur de température TMP 36



Thermistance CTN

- **Protocole de communication:**

. Firmata Express

Rappels :

. Firmata Express

Pour contrôler l'Arduino via le protocole de communication Firmata Express, le programme Python utilise la bibliothèque **"pymata-express"** (le sketch **"Firmata Express"** doit être téléversé dans la mémoire de l'Arduino au préalable).

Toutes les fonctions utiles à l'utilisation de **"Pymata-express"** (fonction de déclaration des entrées et sorties, de lectures, d'écritures…) ont été regroupées dans un fichier Python nommé **"PymataExpressDef.Py"** que l'on peut importer dans tous les programmes, à condition que le fichier des fonctions soit dans le même dossier que le fichier du programme, avec l'instruction :

from PymataExpressDef import *

```
import asyncio


########## DEFINITION D'UNE BOUCLE ASYNCIO ##########

loop = asyncio.get_event_loop()


########## GESTION DES ENTREES NUMERIQUES ##########

# DECLARATION D'UNE BROCHE EN ENTREE NUMERIQUE

def Set_DigitalInput_Pin(board, pin):
    loop.run_until_complete(board.set_pin_mode_digital_input(pin))

# LECTURE DE L'ETAT LOGIQUE D'UNE BROCHE DECLAREE EN ENTREE NUMERIQUE

def Digital_Read(board, pin):
    value = loop.run_until_complete(board.digital_read(pin))
    return value[0]


########## GESTION DES SORTIES NUMERIQUES ##########

# DECLARATION D'UNE BROCHE EN SORTIE NUMERIQUE

def Set_DigitalOutput_Pin(board, pin):
    loop.run_until_complete(board.set_pin_mode_digital_output(pin))

# MODIFICATION DE L'ETAT LOGIQUE D'UNE SORTIE NUMERIQUE

def Digital_Write(board, pin, val):
    loop.run_until_complete(board.digital_write(pin, val))


########## GESTION DES ENTREES ANALOGIQUES ##########

# DECLARATION D'UNE BROCHE EN ENTREE ANALOGIQUE

def Set_AnalogInput_Pin(board,pin):
    loop.run_until_complete(board.set_pin_mode_analog_input(pin))

# LECTURE DE LA VALEUR DE L'ENTREE ANALOGIQUE

def Analog_Read(board, pin):
    value = loop.run_until_complete(board.analog_read(pin))
    return value[0]
```

```
########## GESTION DES SORTIES ANALOGIQUES ##########

# DECLARATION D'UNE BROCHE EN SORTIE ANALOGIQUE

def Set_AnalogOutput_Pin(board,pin):
    loop.run_until_complete(board.set_pin_mode_pwm(pin))

# MODIFICATION DE LA VALEUR D'UNE SORTIE ANALOGIQUE

def Analog_Write(board, pin, val):
    loop.run_until_complete(board.analog_write(pin, val))


########## GESTION DU SON ##########

# DECLARATION D'UNE BROCHE EN MODE TONE

def Set_Tone_Pin(board,pin):
    loop.run_until_complete(board.set_pin_mode_tone(pin))

# EMISSION SONORE

def Tone(board,pin,freq,duration):
    if duration>0:
        loop.run_until_complete(board.play_tone(pin, freq, duration))
    else:
        loop.run_until_complete(board.play_tone_continuously(pin, freq))

# ARRET DE L'EMISSION SONORE

def No_Tone(board,pin):
    loop.run_until_complete(board.play_tone_off(pin))


########## DECONNEXION DE L'ARDUINO ##########

def Arduino_Exit(board):
    loop.run_until_complete(board.shutdown())
```

. Connexion à l'Arduino avec le protocole de communication "Firmata Express"

Dans les programmes de contrôle de l'Arduino par le protocole de communication "Firmata Express", le module "**ConnectToArduino.py**", contenant les fonctions de connexion avec l'Arduino, sera importé.

La connexion se déroulera en 2 étapes :

- Appel de la fonction de sélection du port COM du module "**ConnectToArduino.py**":

  PortComArduino = SelectPortCOM()

  Le nombre de port COM disponible est alors déterminé :

  PortsCOM = list(serial.tools.list_ports.comports()

  –> si nombre de port COM = 0 : message d'erreur,

  –> si nombre de port COM = 1 : sélection de ce port COM pour la connexion,

  –> si nombre de port COM > 1 : L'utilisateur doit sélectionner le bon port COM.

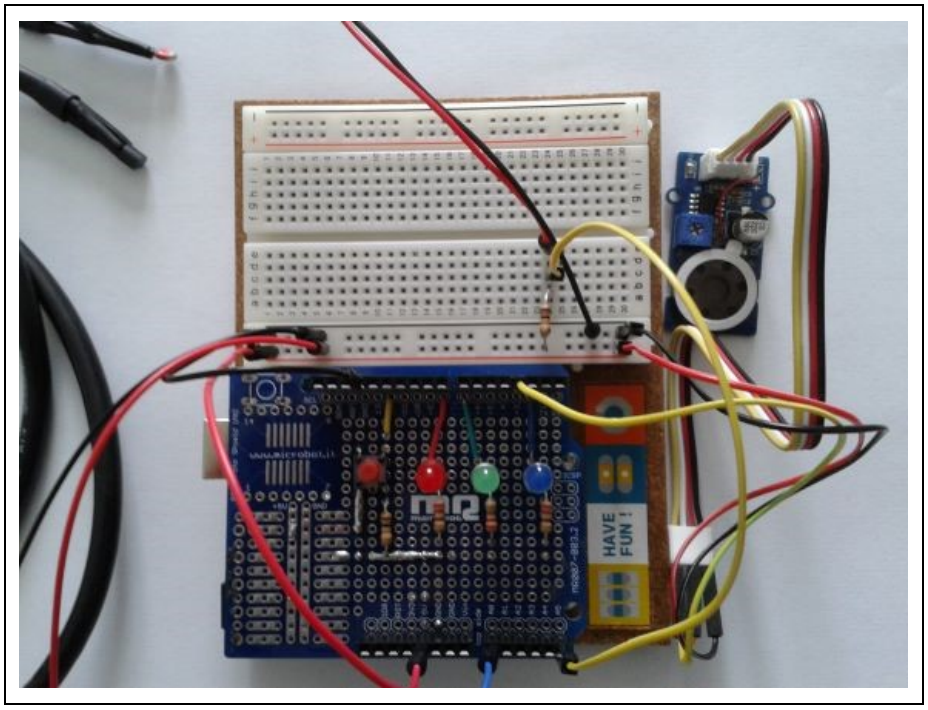
```
def SelectPortCOM():

    Nport=[]
    PortsCOM = list(serial.tools.list_ports.comports())
    for port_numero, description, address in PortsCOM:
        Nport.append(port_numero)


    if len(PortsCOM)== 0:
        print("Aucune carte Arduino n'a été détectée!")
        saisie = ""
        while saisie != "q":
            saisie = str(input("Entrez 'q' pour quitter: "))
        sys.exit()

    elif len(PortsCOM)== 1:
        PortComArduino = Nport[0]

    else:
        print("Liste des ports COM disponibles:\n")
        for i in range(len(PortsCOM)):
            print(i+1, ": ", PortsCOM[i])
        ChoixPort=False
        while ChoixPort==False:
            Choix = input("\nVeuillez indiquer le numéro du port de la carte Arduino:")
            try:
                Choix = int(Choix)
                assert Choix >= 1 and Choix <= len(PortsCOM)
                ChoixPort = True
            except AssertionError:
                print("Le numéro indiqué n'est pas entre 1 et", len(PortsCOM) ,"!")
                ChoixPort = False
            except:
                print("Vous n'avez pas saisi un numéro entre 1 et", len(PortsCOM) ,"!")
        PortComArduino = Nport[Choix-1]

    return PortComArduino
```

- Tentative d'ouverture du port COM sélectionné (**PortComArduino**) et de connexion à l'Arduino via le protocole **"Firmata Express"** avec la fonction **"OpenPortCom"** du module **"ConnectToArduino.py"** :

board = OpenPortCom(PortComArduino)

```
def OpenPortCom(PortCom):

    try:
        board = PymataExpress(com_port = PortCom)

    except:
        AffichMessageErreur(PortCom)

    else:
        return board


def AffichMessageErreur(PortCom):

    print("Un problème s'est produit à l'ouverture du port.\n"
        "Vérifiez que le port utilisé par la carte Arduino est bien "+ PortCom + ",\n" +
        "et que le protocole de communication FIRMATA EXPRESS a bien été téléversé.\n")
    saisie = ""
    while saisie != "q":
        saisie = str(input("Entrez 'q' pour quitter: "))
    sys.exit()
```

- Activité 1 : Mesure de températures avec un capteur TMP 36 ou LM 35



. Objectif

Dans cette activité, nous allons mesurer une température à l'aide d'un capteur TMP 36 ou LM 35.

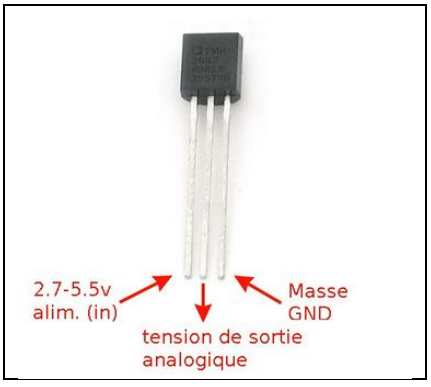
Le principe de fonctionnement de ces capteurs repose sur la dépendance à la température de la caractéristique courant-tension des diodes à jonction au silicium qui sont un des composants du capteur.

Pour un courant fixe, la tension aux bornes d'une diode au silicium varie en fonction de la température.

En amplifiant le changement de tension, ces capteurs génèrent un signal analogique linéairement proportionnel à la température.

. Capteur de température TMP 36

Le capteur dispose de 3 broches :



Caractéristiques :

. tension de fonctionnement : + 2,7 V à + 5,5 V

. sortie calibrée en °C (**10 mV / °C**)

. plage de mesure : −40 ° C à + 125 ° C (fonctionne jusqu'à 150 °C)

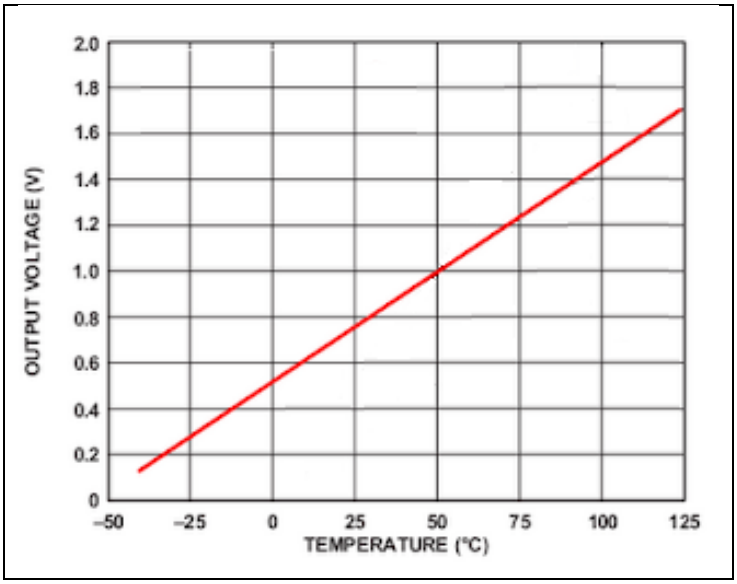
. précision : ±1 ° C à + 25 ° C (± 2 ° C pour des températures en dehors de la plage −40 ° C à + 125 ° C)

. Tension de sortie: 0.1V (-40°C) à 2.0V (150°C)

. Courant de charge: 0.05 mA

Le TMP36 permet de mesurer des températures négatives.

Le 0°C est placé à un offset de 500 mV :



Ainsi, toute mesure inférieure à 500 mv correspondra à une température négative.

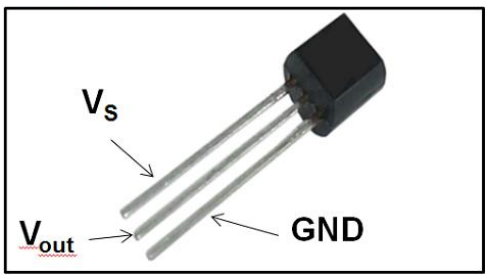
La conversion de la tension de sortie du capteur en température en °C est alors :

T (en °C) = (Tension de sortie (en mV) - 500) / 10

Soit : **T (en °C) = (Tension de sortie (en V) - 0,5) * 100**

. Capteur de température LM 35

Le capteur LM 35, fabriqué par Texas Instrument, dispose également de 3 broches :



Caractéristiques :

. Tension de fonctionnement : + 4 V à + 30 V

. Sortie calibrée en °C (**10 mV / °C**)

. Plage de mesure : −55 ° C à + 150 ° C

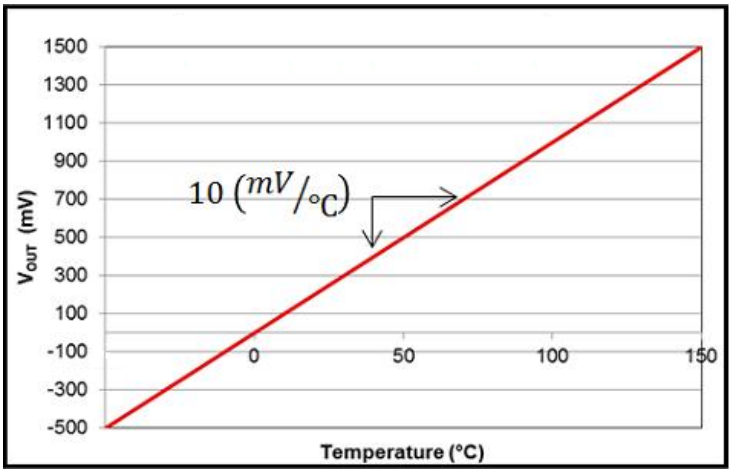
. Précision : +/-0.5°C à 25°C et +/-1°C à -55°C ou +150°C

. Tension de sortie : -0.55V (-55°C) à 1,5V (150°C)

. Courant de charge : 0.06 mA

Avec le capteur LM35, dans le cas de températures en dessous de 0°C, la tension de sortie est négative (-10°C équivaut à -0,1 V).

De fait, avec un Arduino, il n'est possible de mesurer que des températures positives car la tension appliquée sur une entrée analogique du microcontrôleur doit absolument être entre 0 et 5V.



La conversion de la tension de sortie du capteur en température en °C est alors :

T (en °C) = Tension de sortie (en mV) / 10

Soit : **T (en °C) = Tension de sortie (en V) * 100**

. Le programme

Le code de l'activité, en Python ou en langage Arduino, lit la valeur de la broche A0 (nombre entier entre 0 et 1023), convertit cette valeur en tension en V (nombre décimal entre 0 et 5 V) et affiche la température correspondante dans dans la fenêtre Python Shell ou le moniteur série.

Par défaut, le code est prévu pour un capteur TMP 36, mais le calcul de la température avec un LM 35 étant en commentaire, il suffit de modifier le code pour mesurer une température avec ce capteur.

. Programme en Python (Projet5\Activity1\PY\Activity1.py)

```
# Importations des librairies et définition des fonctions

from PymataExpressDef import *
from ConnectToArduino import *
import time

# Déclaration des constantes et variables

PinSensor = 0
sensorVal = 0
tension = 0
temperature = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties


Set_AnalogInput_Pin(board, PinSensor)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter.")


# Boucle principale du programme

while True:
    try:
        sensorVal = Analog_Read(board, PinSensor)
        tension = (sensorVal/1023)*5

        # Capteur TMP 36
        temperature = (tension - 0.5) * 100;

        # Capteur LM 35
        # temperature = tension * 100;

        print(sensorVal, " ; ", round(tension,2), " ; ", round(temperature,1))
        time.sleep(0.1)

    except KeyboardInterrupt:
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module **"ConnectToArduino.py",** contenant les fonctions de connexion à l'Arduino via le protocole **"Firmata Express"**,

. Le module "**PymataExpressDef.Py**" regroupant toutes les fonctions utiles à l'utilisation de " **Pymata-express** " (fonction de déclaration des entrées et sorties, de lectures, d'écritures…),

. La bibliothèque **"time"** pour la gestion des temps de pause.

- Déclaration des constantes et variables :

. **PinSensor = 0** (cst correspondant au n° de la broche A0 sur laquelle le capteur de température est connecté)

. **ValSensor = 0** (variable pour stocker la valeur de la broche du capteur de température)

. **tension = 0** (variable correspondant à la valeur de la tension en V de la broche du capteur de température)

. **temperature = 0** (variable correspondant à la température en °C calculée à partir de la valeur de la broche du capteur)

. **PortComArduino** (port COM sur lequel l'Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM, tentative d'ouverture du port COM sélectionné et connexion à l'Arduino :

PortComArduino = SelectPortCOM()

board = OpenPortCom(PortComArduino)

. Si la connexion à l'Arduino est réussie :

- Déclaration de la broche du capteur de température en entrée analogique :

Set_AnalogInput_Pin(board, PinSensor)

- Boucle principale du programme (boucle "while True") :

. Lecture de la valeur de la broche du capteur de température :

sensorVal = Analog_Read(board, PinSensor)

. Calcul de la tension en V de la broche du capteur de température :

tension = (sensorVal/1023)*5

. Calcul de la température en °C correspondante (exemple pour un capteur TMP 36) :

temperature = (tension - 0.5) * 100

. Affichage de ces 3 valeurs dans la fenêtre Python Shell.

. Attente de 100 ms avant une nouvelle mesure :

time.sleep(0.1)

- Fin du programme en appuyant sur Ctrl-C :

→ Le port COM est fermé.

Résultats dans la fenêtre Python Shell :

```
Pymata Express Version 1.4
Copyright (c) 2018-2019 Alan Yorinks All rights reserved.

Opening COM3 ...
Waiting 4 seconds for the Arduino To Reset.

Arduino found and connected to COM3

Retrieving Arduino Firmware ID...
Arduino Firmware ID: 2.5 FirmataExpress.ino
Auto-discovery complete. Found 20 Digital Pins and 6 Analog Pins


Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter.
140  ;  0.68  ;  18.4
140  ;  0.68  ;  18.4
139  ;  0.68  ;  17.9
140  ;  0.68  ;  18.4
139  ;  0.68  ;  17.9
140  ;  0.68  ;  18.4
139  ;  0.68  ;  17.9
140  ;  0.68  ;  18.4
140  ;  0.68  ;  18.4
139  ;  0.68  ;  17.9
140  ;  0.68  ;  18.4
>>>
```

. Programme en langage Arduino (Projet5\Activity1\INO\Activity1.ino)

Activity1

```
// Déclaration des constantes et variables

int sensorVal = 0;
float tension = 0.0;
float temperature = 0.0;


// Initialisation des entrées et sorties

void setup() {
  Serial.begin(9600);
  Serial.print("Valeur du capteur");
  Serial.print(" ; Tension en Volt");
  Serial.println(" ; Temperature en degre Celsius:");
}


// Fonction principale en boucle

void loop() {
  sensorVal = analogRead(A0);
  tension = (sensorVal/1023.0)*5.0;

  // Capteur TMP 36
  temperature = (tension - 0.5) * 100;

  // Capteur LM 35
  //temperature = tension * 100;


  Serial.print(sensorVal);
  Serial.print(" ; ");
  Serial.print(tension,2);
  Serial.print(" ; ");
  Serial.println(temperature,1);
  delay(100);
}
```

Déroulement du programme :

- Déclaration des constantes et variables :

  . **int sensorVal = 0** (variable pour stocker la valeur de la broche A0)

  . **float tension = 0.0** (variable pour stocker la valeur en V de la tension correspondante à la valeur de la broche A0)

  . **float temperature = 0.0** (variable pour stocker le résultat du calcul de la température)

- Initialisation des entrées et sorties :

. La liaison série est établie à un débit de 9600 bauds,

. L'entête des mesures effectuées est affiché dans le moniteur série.

- Fonction principale en boucle :

. Lecture de la valeur de la broche A0 : **sensorVal = analogRead(A0)**

. Calcul de la tension en V correspondante :

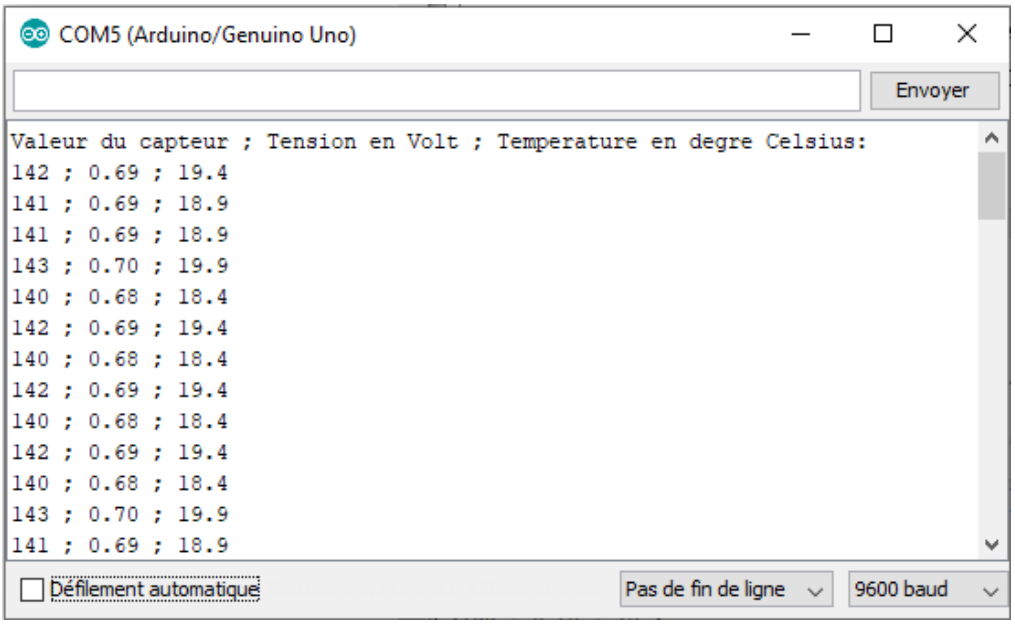
tension = (sensorVal/1023.0)*5.0

. Calcul de la température en °C correspondante :

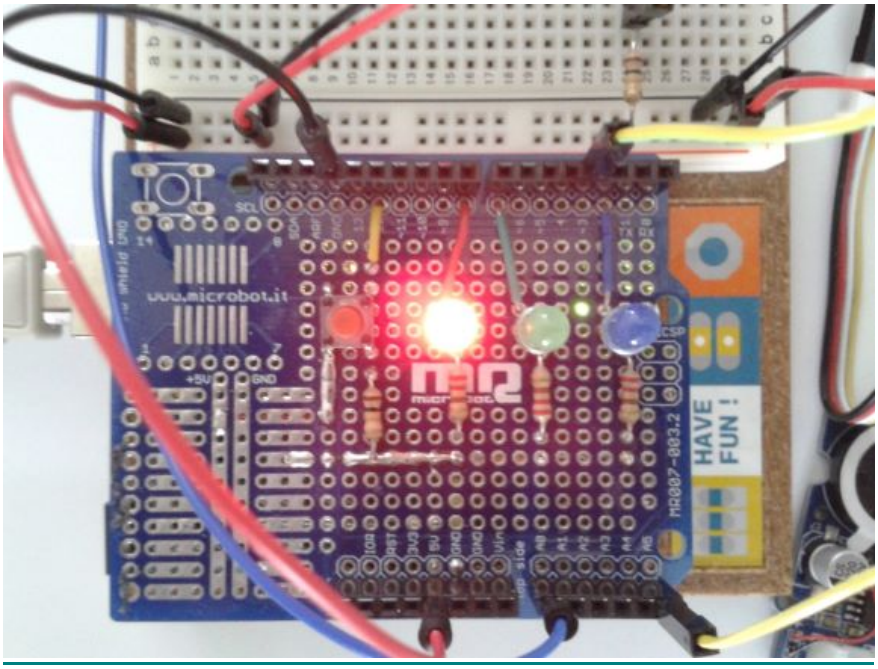
- Pour un TMP 36 : **temperature = (tension - 0.5) * 100**
- Pour un LM 35 : **temperature = tension * 100**

. Affichage des variables sensorVal, tension et temperature dans le moniteur série.

Résultats dans le moniteur série :



- Activité 2 : Alarme sonore par dépassement de température



. Objectif

L'objectif de cette activité est de réaliser une alarme sonore et visuelle qui se déclenchera lorsque la température mesurée par le capteur TMP 36 ou LM 35 de notre circuit d'étude est supérieure à une valeur seuil à définir, permettant, par exemple, de prévenir un utilisateur du dépassement de la température d'utilisation d'un matériel.

Il suffit pour cela de reprendre le programme de l'activité précédente, auquel nous allons ajouter le code de l'alarme sonore et visuelle déjà vu.

Les mesures de température commencent après un appui sur le bouton poussoir et sont arrêtées en appuyant de nouveau sur celui-ci.

. Le programme

Le code de l'activité en Python ou en langage Arduino pourra être modifié pour voir l'influence des variables (seuil de température, fréquence de l'onde sonore, durée d'émission, durée de silence).

. Programme en Python (Projet5\Activity2\PY\Activity2.py)

```
# Importations des librairies et définition des fonctions

from PymataExpressDef import *
from ConnectToArduino import *
import time


# Déclaration des constantes et variables

PinSensor=0
PinButton= 12
PinLed = 8
PinTone = 3

ValSensor = 0
tension = 0.0
Temp = 0.0
OldTemp = 0.0
TempAlarme = 22.0

ValButton = 0
OldValButton = 0
State = 0
OldState = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

Set_AnalogInput_Pin(board, PinSensor)
Set_DigitalOutput_Pin(board, PinLed)
Set_DigitalInput_Pin(board, PinButton)
Set_Tone_Pin(board,PinTone)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter.\n")
print("Appuyez sur le bouton poussoir pour commencer les mesures.\n");


# Boucle principale du programme

while True:
    try:
        ValButton = Digital_Read(board, PinButton)
        time.sleep(0.01)
        if ValButton == 1 and OldValButton == 0:
            State=1-State

        OldValButton = ValButton;
```

```
        if State==1:
            if OldState == 0:
               print("\nMesure de la température en cours.\n")
               print("Température en degré Celsius:")
               OldState=1

            sensorVal = Analog_Read(board, PinSensor)
            tension = (sensorVal/1023)*5

            # Capteur TMP 36
            Temp = (tension - 0.5) * 100;

            # Capteur LM 35
            # Temp = tension * 100;

            if OldTemp != Temp:
                print(round(Temp,1))
                OldTemp = Temp

            if Temp > TempAlarme:
                Digital_Write(board, PinLed, 1)
                Tone(board, PinTone,440, 0)
                time.sleep(0.1)
                Digital_Write(board, PinLed, 0)
                No_Tone(board, PinTone)
                time.sleep(0.1)
            else:
                Digital_Write(board, PinLed, 0)
                No_Tone(board, PinTone)

            time.sleep(0.1)

        else:
            if OldState == 1:
                print("Fin des mesures.\n")
                OldState = 0

    except KeyboardInterrupt:
        Digital_Write(board, PinLed, 0)
        No_Tone(board, PinTone)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module **”ConnectToArduino.py”,** contenant les fonctions de connexion à l’Arduino via le protocole **”Firmata Express”**,

. Le module "**PymataExpressDef.Py**" regroupant toutes les fonctions utiles à l’utilisation de " **Pymata-express** " (fonction de déclaration des entrées et sorties, de lectures, d’écritures…),

. La bibliothèque **"time"** pour la gestion des temps de pause.

- Déclaration des constantes et variables :

. **PinSensor = 0** (cst correspondant au n° de la broche A0 sur laquelle le capteur de température est connecté)

. **ValSensor = 0** (variable pour stocker la valeur de la broche du capteur de température)

. **tension = 0** (variable correspondant à la valeur de la tension en V de la broche du capteur de température)

. **Temp = 0** (variable correspondant à la température en °C calculée à partir de la valeur de la broche du capteur)

. **OldTemp = 0** (variable correspondant à la température en °C calculée précédemment)

. **TempAlarme = 22.0** (cst correspondant à la température du seuil de déclenchement de l'alarme)

. **PinButton = 12** (cst correspondant au n° de la broche sur laquelle le bouton poussoir est connecté)

. **PinLed = 8** (cst correspondant au n° de la broche sur laquelle la DEL est connectée)

. **PinTone = 3** (cst correspondant au n° de la broche sur laquelle le buzzer est connecté)

. **ValButton = 0** (variable pour stocker la valeur de l'état logique de la broche du bouton poussoir)

. **OldValButton = 0** (variable pour stocker la valeur précédente de l'état logique de la broche du bouton poussoir)

. **State=0** (variable correspondant à l'action à effectuer)

. **OldState = 0** (variable pour stocker la valeur précédente de la variable **state**)

. **PortComArduino** (port COM sur lequel l'Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM, tentative d'ouverture du port COM sélectionné et connexion à l'Arduino :

PortComArduino = SelectPortCOM()

board = OpenPortCom(PortComArduino)

. Si la connexion à l'Arduino est réussie :

- Déclaration de la broche du capteur de température en entrée analogique :

Set_AnalogInput_Pin(board, PinSensor)

- Déclaration de la broche de la DEL en sortie numérique :

Set_DigitalOutput_Pin(board, PinLed)

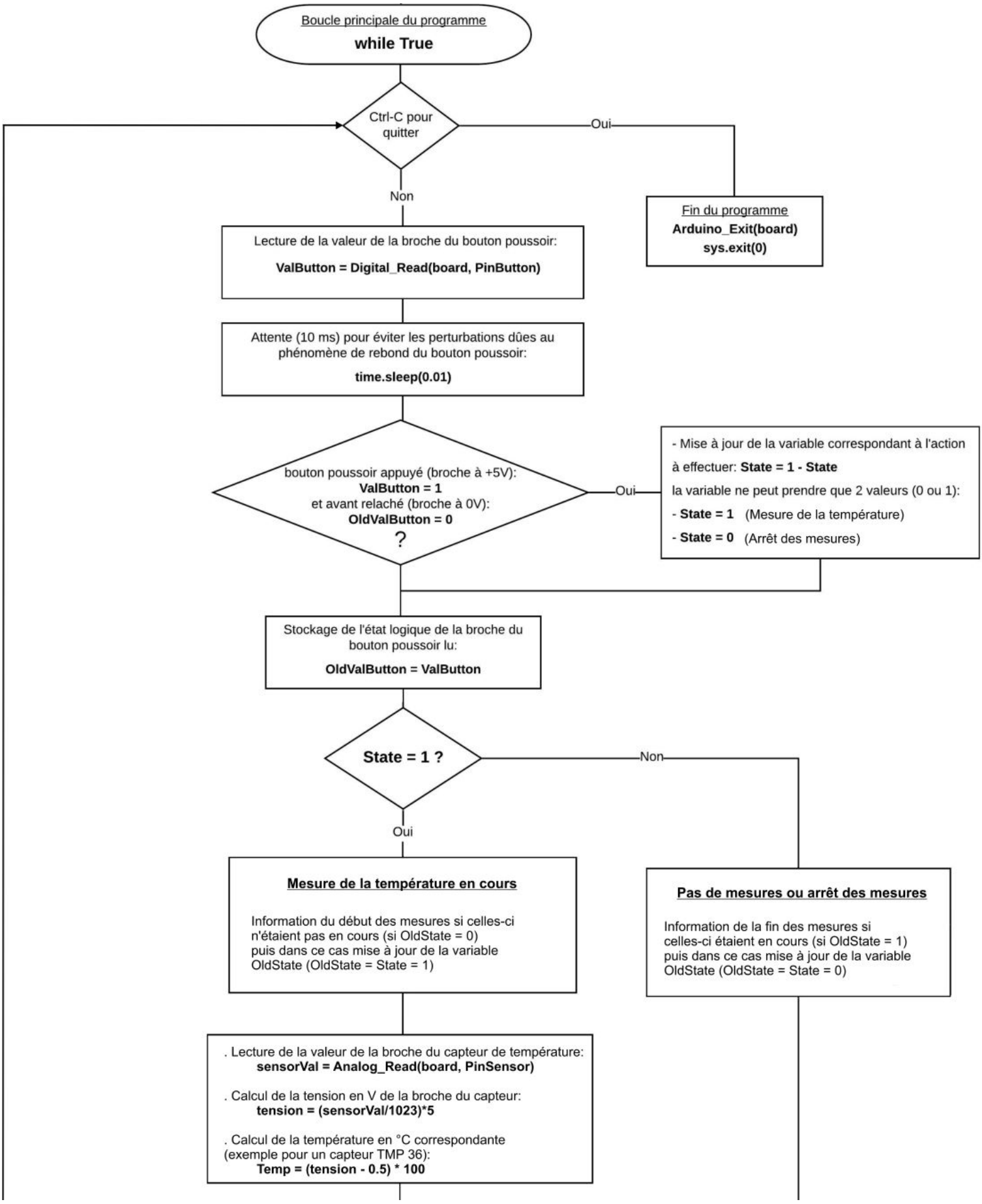
- Déclaration de la broche du bouton poussoir en entrée numérique :

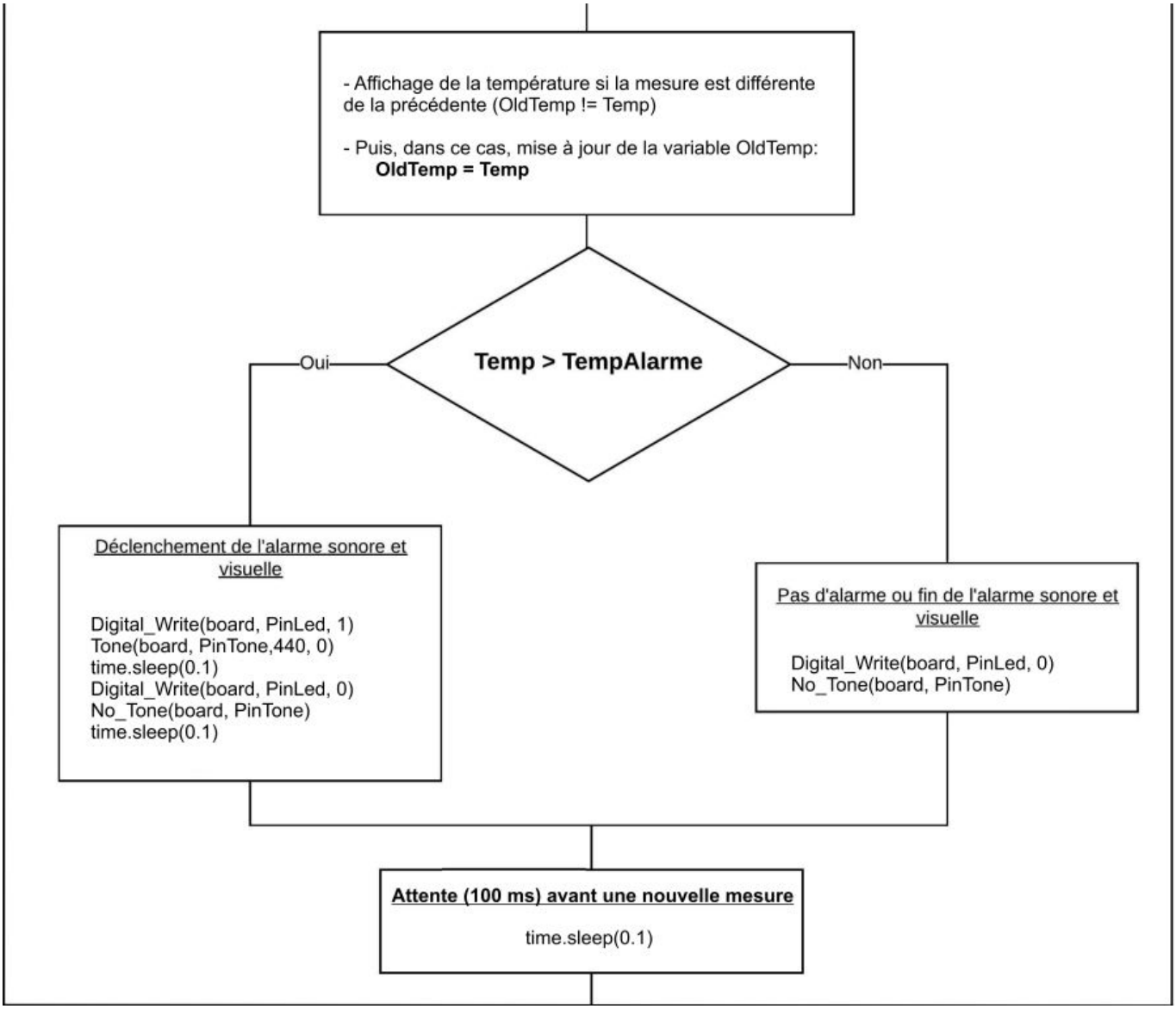
Set_DigitalInput_Pin(board, PinButton)

- Déclaration de la broche du buzzer en mode « Tone » :

Set_Tone_Pin(board,PinTone)

- Boucle principale du programme (boucle "while True") :





Résultats dans la fenêtre Python Shell :

```
Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter.

Appuyez sur le bouton poussoir pour commencer les mesures.


Mesure de la température en cours.

Température en degré Celsius:
17.0
17.4
17.0
17.4
17.0
16.5
17.0
17.4
17.0
Fin des mesures.
```

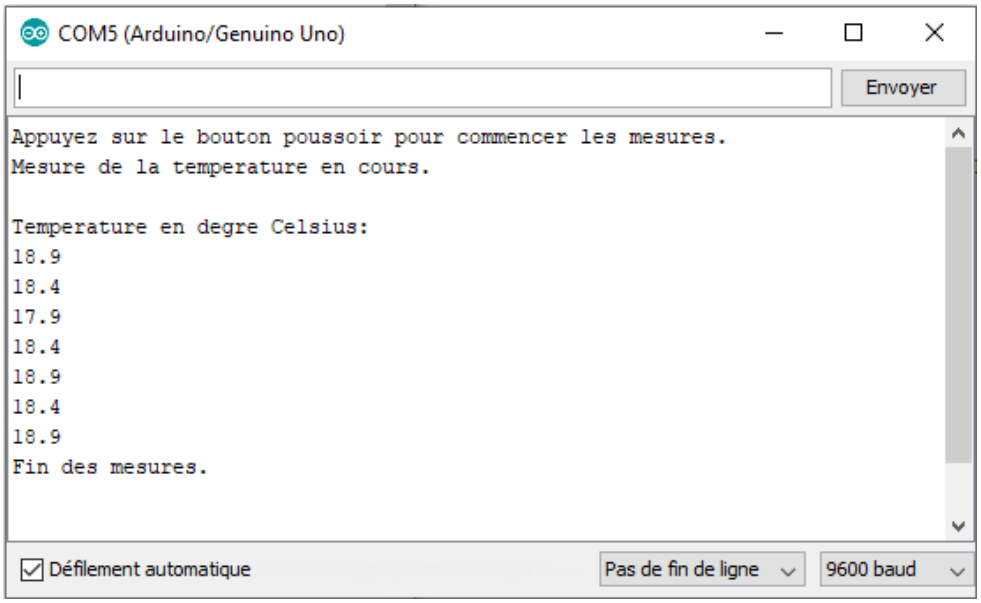
. **Programme en langage Arduino (Projet5\Activity2\INO\Activity2.ino)**

Activity2

```
// Déclaration des constantes et variables

const int PinSensor=0;
const int PinButton= 12;
const int PinLed = 8;
const int PinTone = 3;

int ValSensor = 0;
float tension = 0.0;
float Temp = 0.0;
float OldTemp = 0.0;
float TempAlarme = 25.0;

int ValButton = 0;
int OldValButton = 0;
int State = 0;
int OldState = 0;


// Initialisation des entrées et sorties

void setup() {
  Serial.begin(9600);
  pinMode(PinButton,INPUT);
  pinMode(PinLed,OUTPUT);
  Serial.println("Appuyez sur le bouton poussoir pour commencer les mesures.");
}


// Fonction principale en boucle

void loop() {
  ValButton = digitalRead(PinButton);
delay(10);

if ((ValButton == HIGH)&&(OldValButton == LOW))
  {
  State=1-State;
  }
OldValButton = ValButton;
```

```
if (State==1)
  {
  if (OldState == 0)
  {
   Serial.println("Mesure de la temperature en cours.");
   Serial.println("");
   Serial.println ("Temperature en degre Celsius:");
   OldState=1;
  }
  ValSensor = analogRead(PinSensor);
  tension = (ValSensor/1023.0)*5.0;

  // Capteur TMP 36
  Temp = (tension - 0.5) * 100;

  // Capteur LM 35
  //Temp = tension * 100;

  if (OldTemp != Temp)
  {
    Serial.println(Temp,1);
    OldTemp = Temp;
  }
  if (Temp > TempAlarme)
   {
      digitalWrite(PinLed, HIGH);
      tone(PinTone,440);
      delay(100);
      digitalWrite(PinLed, LOW);
      noTone(PinTone);
      delay(100);
   }
   else
   {
      digitalWrite(PinLed, LOW);
      noTone(PinTone);
   }
  delay(100);
  }
  else
  {
    if (OldState == 1){
    Serial.println("Fin des mesures.");
    OldState = 0;}
  }
}
```

Résultats dans le moniteur série :



Déroulement du programme :

- Déclaration des constantes et variables :

- N° de la broche correspondant au bouton poussoir: **const int PinButton = 12**
- N° de la broche correspondant à la DEL : **const int PinLed = 8**
- N° de la broche correspondant au piezo : **const int PinTone = 3**
- N° de la broche correspondant au capteur de température : **const int PinSensor = 0**
- Variable pour stocker la valeur de la broche du bouton poussoir: **int ValButton = 0**
- Variable pour stocker l'ancienne valeur de la broche du bouton poussoir: **int OldValButton = 0**
- Variable correspondant à l'action à effectuer: **int State = 0**
- Variable pour stocker l'ancienne valeur de la variable correspondant à l'action à effectuer: **int OldState = 0**
- Variable pour stocker la valeur de la broche du capteur de température: **int ValSensor = 0**
- Variable pour stocker la valeur en V de la tension correspondante à la valeur de la broche du capteur : **float tension = 0.0**
- Variable correspondant à la température calculée à partir de la valeur de la broche du capteur: **float Temp = 0.0**
- Variable correspondant à la température mesurée précédemment: **float OldTemp = 0.0**
- Variable correspondant à la température du seuil de déclenchement de l'alarme: **float TempAlarme = 25.0**

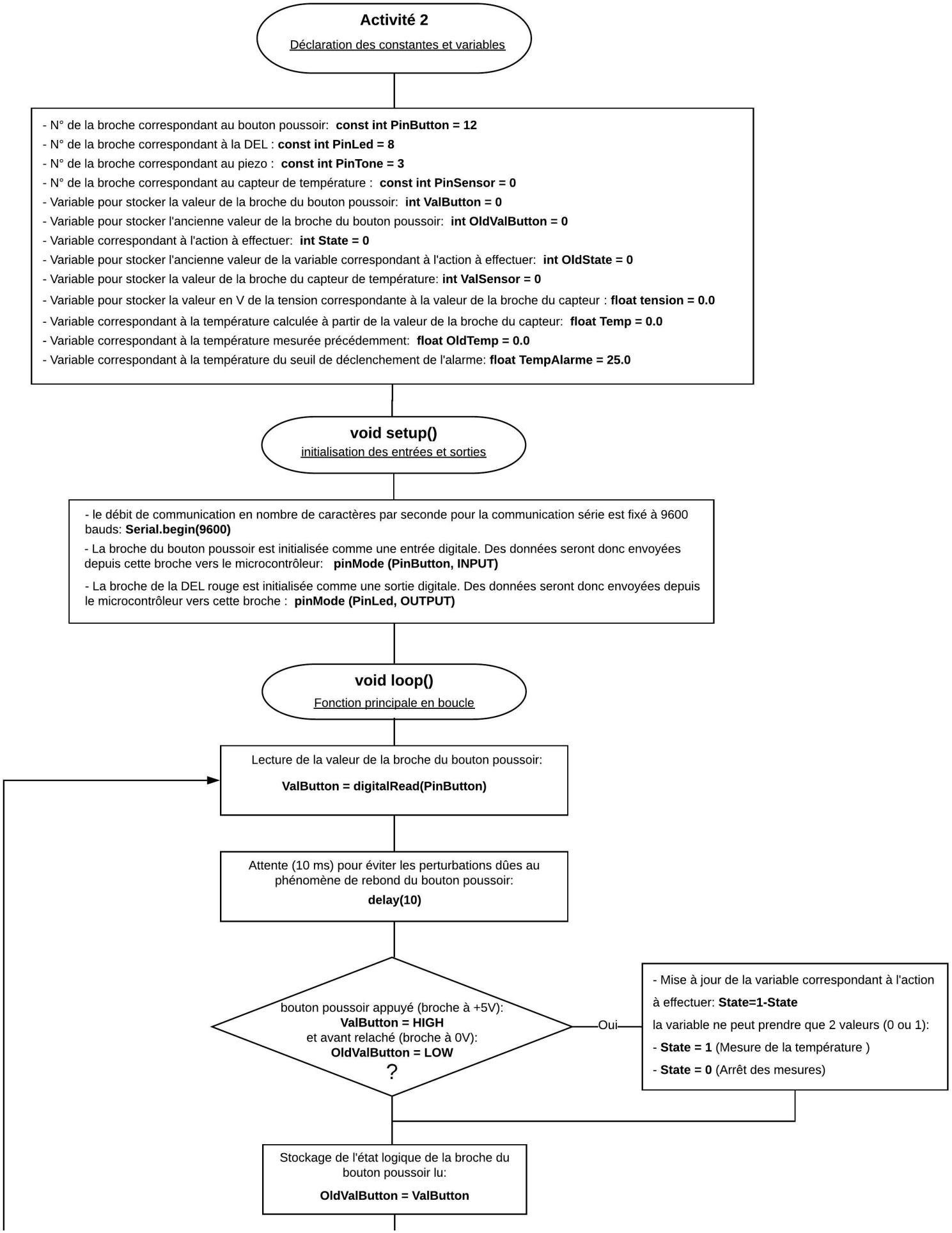
- Initialisation des entrées et sorties :

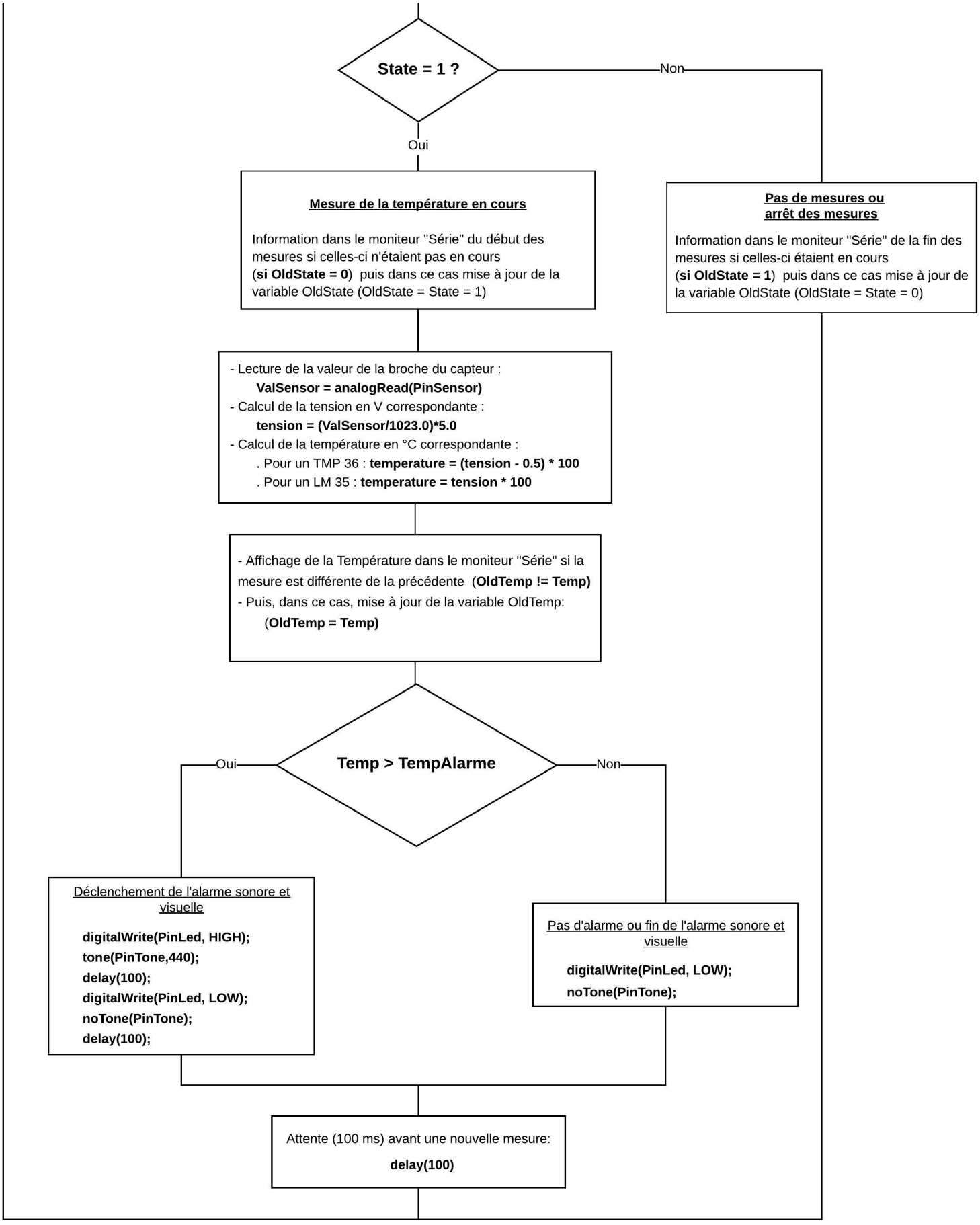
- le débit de communication en nombre de caractères par seconde pour la communication série est fixé à 9600 bauds: **Serial.begin(9600)**

- La broche du bouton poussoir est initialisée comme une entrée digitale. Des données seront donc envoyées depuis cette broche vers le microcontrôleur: **pinMode (PinButton, INPUT)**

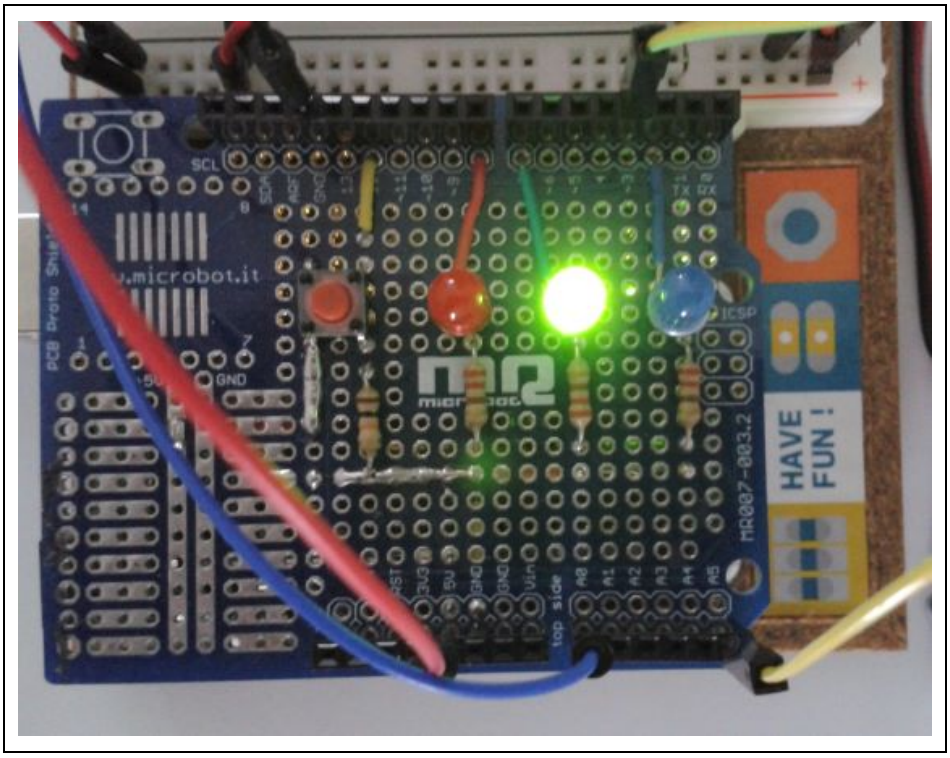
- La broche de la DEL rouge est initialisée comme une sortie digitale. Des données seront donc envoyées depuis le microcontrôleur vers cette broche : **pinMode (PinLed, OUTPUT)**

- Fonction principale en boucle :




State = 1 ?
Non
Oui
Mesure de la température en cours
Information dans le moniteur "Série" du début des mesures si celles-ci n'étaient pas en cours (si OldState = 0) puis dans ce cas mise à jour de la variable OldState (OldState = State = 1)
Pas de mesures ou arrêt des mesures
Information dans le moniteur "Série" de la fin des mesures si celles-ci étaient en cours (si OldState = 1) puis dans ce cas mise à jour de la variable OldState (OldState = State = 0)
- Lecture de la valeur de la broche du capteur :
ValSensor = analogRead(PinSensor)
- Calcul de la tension en V correspondante :
tension = (ValSensor/1023.0)*5.0
- Calcul de la température en °C correspondante :
. Pour un TMP 36 : temperature = (tension - 0.5) * 100
. Pour un LM 35 : temperature = tension * 100
- Affichage de la Température dans le moniteur "Série" si la mesure est différente de la précédente (OldTemp != Temp)
- Puis, dans ce cas, mise à jour de la variable OldTemp:
(OldTemp = Temp)
Temp > TempAlarme
Oui
Non
Déclenchement de l'alarme sonore et visuelle
digitalWrite(PinLed, HIGH);
tone(PinTone,440);
delay(100);
digitalWrite(PinLed, LOW);
noTone(PinTone);
delay(100);
Pas d'alarme ou fin de l'alarme sonore et visuelle
digitalWrite(PinLed, LOW);
noTone(PinTone);
Attente (100 ms) avant une nouvelle mesure:
delay(100)

- Activité 3 : Thermomètre à diodes électroluminescentes



. Objectif

Dans cette activité, nous allons utiliser les DELS rouge, verte et bleue du circuit d'étude afin de visualiser la zone dans laquelle se situe la température mesurée ($T_{mesurée}$) par un capteur TMP 36 ou LM 35 par rapport à une valeur de référence (T_{ref}) et un écart de température (ΔT) à définir :

- si $T_{mesurée} < T_{ref} - \Delta T$: La DEL bleue est allumée,
- si $T_{mesurée} > T_{ref} + \Delta T$: La DEL rouge est allumée,
- $T_{ref} - \Delta T < T_{mesurée} < T_{ref} + \Delta T$: La DEL verte est allumée.

. Le programme

Le code de l'activité en Python ou en langage Arduino pourra être modifié pour voir l'influence des variables (température de référence, écart de température).

. Programme en Python (**Projet5\Activity3\PY\Activity3.py**)

```
# Importations des librairies et définition des fonctions

from PymataExpressDef import *
from ConnectToArduino import *
import time


# Déclaration des constantes et variables

PinSensor=0
PinButton= 12
PinLedR = 8
PinLedV = 7
PinLedB = 2
TempRef = 20
DT = 1

ValSensor = 0
tension = 0.0
Temp = 0.0
OldTemp = 0.0


ValButton = 0
OldValButton = 0
State = 0
OldState = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

Set_AnalogInput_Pin(board, PinSensor)
Set_DigitalOutput_Pin(board, PinLedR)
Set_DigitalOutput_Pin(board, PinLedV)
Set_DigitalOutput_Pin(board, PinLedB)
Set_DigitalInput_Pin(board, PinButton)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter.\n")
print("Appuyez sur le bouton poussoir pour commencer les mesures.\n");


# Boucle principale du programme

while True:
    try:
        ValButton = Digital_Read(board, PinButton)
        time.sleep(0.01)
        if ValButton == 1 and OldValButton == 0:
            State=1-State

        OldValButton = ValButton;
```

```
        if State==1:
            if OldState == 0:
               print("\nMesure de la température en cours.\n")
               print("Température en degré Celsius:")
               OldState=1

            sensorVal = Analog_Read(board, PinSensor)
            tension = (sensorVal/1023)*5

            # Capteur TMP 36
            Temp = (tension - 0.5) * 100;

            # Capteur LM 35
            # Temp = tension * 100;

            if OldTemp != Temp:
                print(round(Temp,1))
                OldTemp = Temp

            if (Temp > TempRef - DT) and (Temp < TempRef + DT):
                Digital_Write(board, PinLedR, 0)
                Digital_Write(board, PinLedV, 1)
                Digital_Write(board, PinLedB, 0)

            if Temp > TempRef + DT:
                Digital_Write(board, PinLedR, 1)
                Digital_Write(board, PinLedV, 0)
                Digital_Write(board, PinLedB, 0)

            if Temp < TempRef - DT:
                Digital_Write(board, PinLedR, 0)
                Digital_Write(board, PinLedV, 0)
                Digital_Write(board, PinLedB, 1)

            time.sleep(0.1)

        else:
            if OldState == 1:
                print("Fin des mesures.\n")
                Digital_Write(board, PinLedR, 0)
                Digital_Write(board, PinLedV, 0)
                Digital_Write(board, PinLedB, 0)
                OldState = 0

    except KeyboardInterrupt:
        Digital_Write(board, PinLedR, 0)
        Digital_Write(board, PinLedV, 0)
        Digital_Write(board, PinLedB, 0)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module **”ConnectToArduino.py”,** contenant les fonctions de connexion à l’Arduino via le protocole **”Firmata Express”**,

. Le module "**PymataExpressDef.Py**" regroupant toutes les fonctions utiles à l’utilisation de " **Pymata-express** " (fonction de déclaration des entrées et sorties, de lectures, d’écritures…),

. La bibliothèque **"time"** pour la gestion des temps de pause.

- Déclaration des constantes et variables :

. **PinSensor = 0** (cst correspondant au n° de la broche A0 sur laquelle le capteur de température est connecté)

. **ValSensor = 0** (variable pour stocker la valeur de la broche du capteur de température)

. **tension = 0** (variable correspondant à la valeur de la tension en V de la broche du capteur de température)

. **Temp = 0** (variable correspondant à la température en °C calculée à partir de la valeur de la broche du capteur)

. **OldTemp = 0** (variable correspondant à la température en °C calculée précédemment)

. **TempRef = 20** (constante correspondant à la température de référence en °C)

. **DT = 1** (cst correspondant à l'écart de température en °C par rapport à la température de référence)

. **PinButton = 12** (cst correspondant au n° de la broche sur laquelle le bouton poussoir est connecté)

. **PinLedR = 8** (cst correspondant au n° de la broche sur laquelle la DEL rouge est connectée)

. **PinLedV = 7** (cst correspondant au n° de la broche sur laquelle la DEL verte est connectée)

. **PinLedB = 2** (cst correspondant au n° de la broche sur laquelle la DEL verte est connectée)

. **ValButton = 0** (variable pour stocker la valeur de l’état logique de la broche du bouton poussoir)

. **OldValButton = 0** (variable pour stocker la valeur précédente de l’état logique de la broche du bouton poussoir)

. **State=0** (variable correspondant à l’action à effectuer)

. **OldState = 0** (variable pour stocker la valeur précédente de la variable **state**)

. **PortComArduino** (port COM sur lequel l’Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM, tentative d’ouverture du port COM sélectionné et connexion à l’Arduino :

PortComArduino = SelectPortCOM()

board = OpenPortCom(PortComArduino)

. Si la connexion à l'Arduino est réussie :

- Déclaration de la broche du capteur de température en entrée analogique :

  Set_AnalogInput_Pin(board, PinSensor)

- Déclaration des broches des DELs en sorties numériques :

  Set_DigitalOutput_Pin(board, PinLedR)

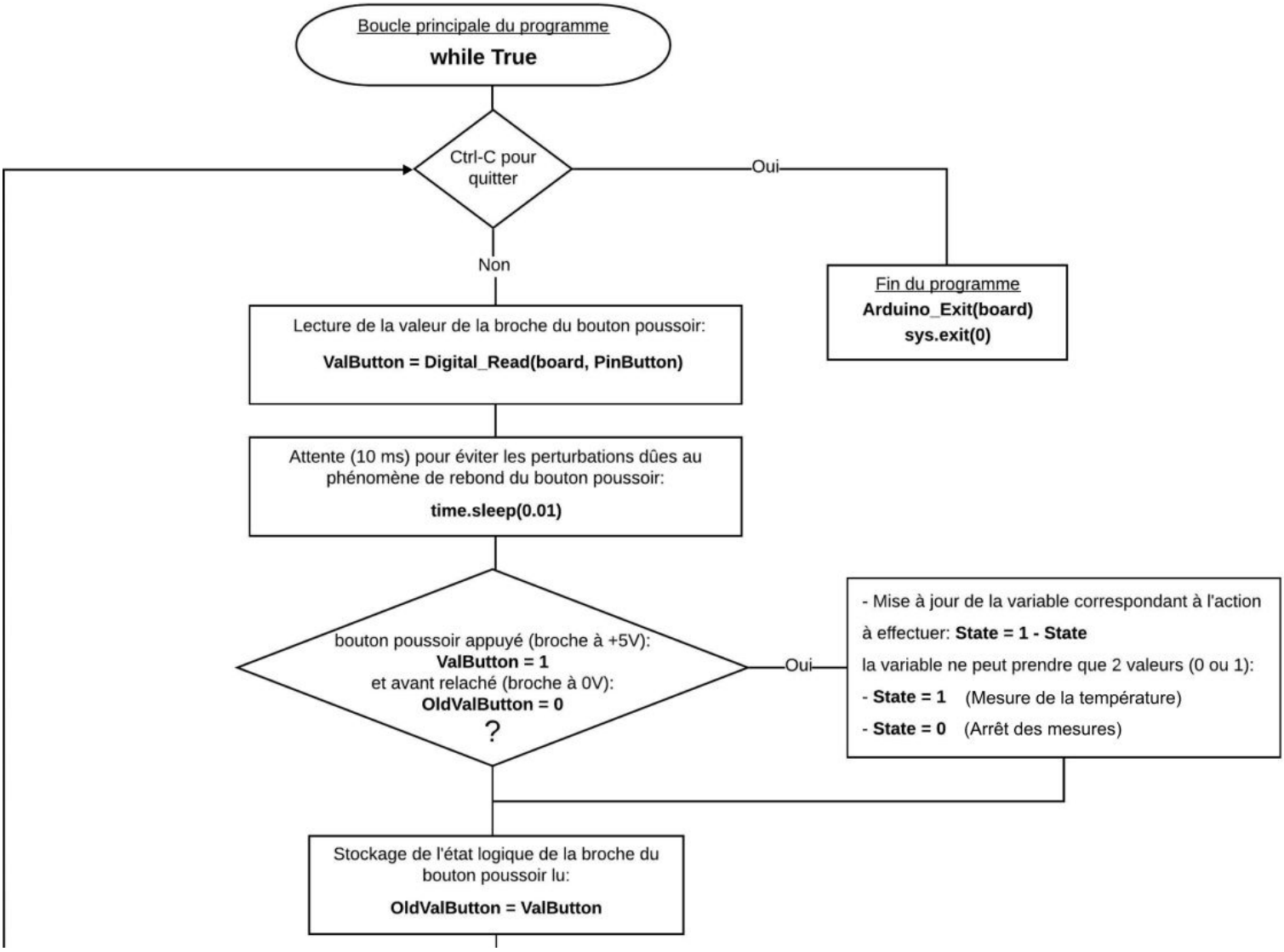
  Set_DigitalOutput_Pin(board, PinLedV)

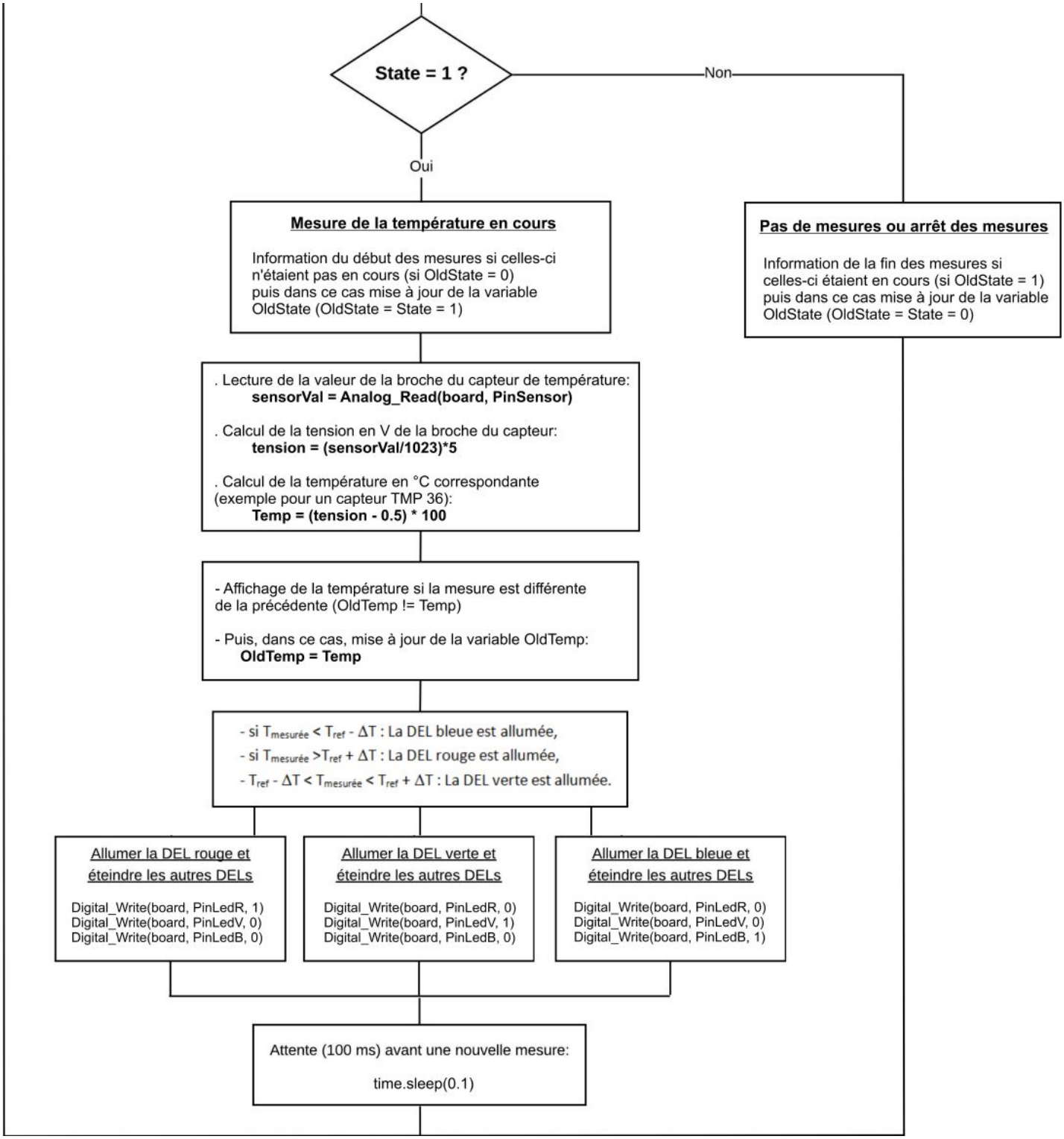
  Set_DigitalOutput_Pin(board, PinLedB)

- Déclaration de la broche du bouton poussoir en entrée numérique :

  Set_DigitalInput_Pin(board, PinButton)

- Boucle principale du programme (boucle "while True") :





Résultats dans la fenêtre Python Shell :

```
Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter.

Appuyez sur le bouton poussoir pour commencer les mesures.


Mesure de la température en cours.

Température en degré Celsius:
17.9
18.4
17.9
Fin des mesures.
```

. **Programme en langage Arduino (Projet5\Activity3\INO\Activity3.ino)**

Activity3.ino

```
// Déclaration des constantes et variables

const int PinSensor=0;
const int PinButton= 12;
const int PinLedR = 8;
const int PinLedV = 7;
const int PinLedB = 2;
const float TempRef = 20.0;
const float DT = 1.0;

int ValSensor = 0;
float tension = 0.0;
float Temp = 0.0;
float OldTemp = 0.0;

int ValButton = 0;
int OldValButton = 0;
int State = 0;
int OldState = 0;


// Initialisation des entrées et sorties

void setup() {
  Serial.begin(9600);
  pinMode(PinButton,INPUT);
  pinMode(PinLedR,OUTPUT);
  pinMode(PinLedV,OUTPUT);
  pinMode(PinLedB,OUTPUT);
  Serial.println("Appuyez sur le bouton poussoir pour commencer les mesures.");
}


 // Fonction principale en boucle

 void loop() {
   ValButton = digitalRead(PinButton);
   delay(10);

   if ((ValButton == HIGH)&&(OldValButton == LOW))
   {
   State=1-State;
   }
   OldValButton = ValButton;
```

```
  if (State==1)
  {
  if (OldState == 0)
  {
   Serial.println("Mesure de la temperature en cours.");
   Serial.println("");
   Serial.println ("Temperature en degre Celsius:");
   OldState=1;
  }
  ValSensor = analogRead(PinSensor);
  tension = (ValSensor/1023.0)*5.0;

  // Capteur TMP 36
  Temp = (tension - 0.5) * 100;

  // Capteur LM 35
  //Temp = tension * 100;

  if (OldTemp != Temp)
  {
    Serial.println(Temp,1);
    OldTemp = Temp;
  }
  if ((Temp > TempRef - DT) && (Temp < TempRef + DT)) {
    digitalWrite(PinLedR, LOW);
    digitalWrite(PinLedV, HIGH);
    digitalWrite(PinLedB, LOW);
  }

  if (Temp > TempRef + DT)  {
    digitalWrite(PinLedR, HIGH);
    digitalWrite(PinLedV, LOW);
    digitalWrite(PinLedB, LOW);
  }

  if (Temp < TempRef - DT)  {
    digitalWrite(PinLedR, LOW);
    digitalWrite(PinLedV, LOW);
    digitalWrite(PinLedB, HIGH);
  }
  delay(100);
  }
  else
  {
   if (OldState == 1){
   Serial.println("Fin des mesures.");
   digitalWrite(PinLedB, LOW);
   digitalWrite(PinLedR, LOW);
   digitalWrite(PinLedV, LOW);
   OldState = 0;}
  }
}
```

Déroulement du programme :

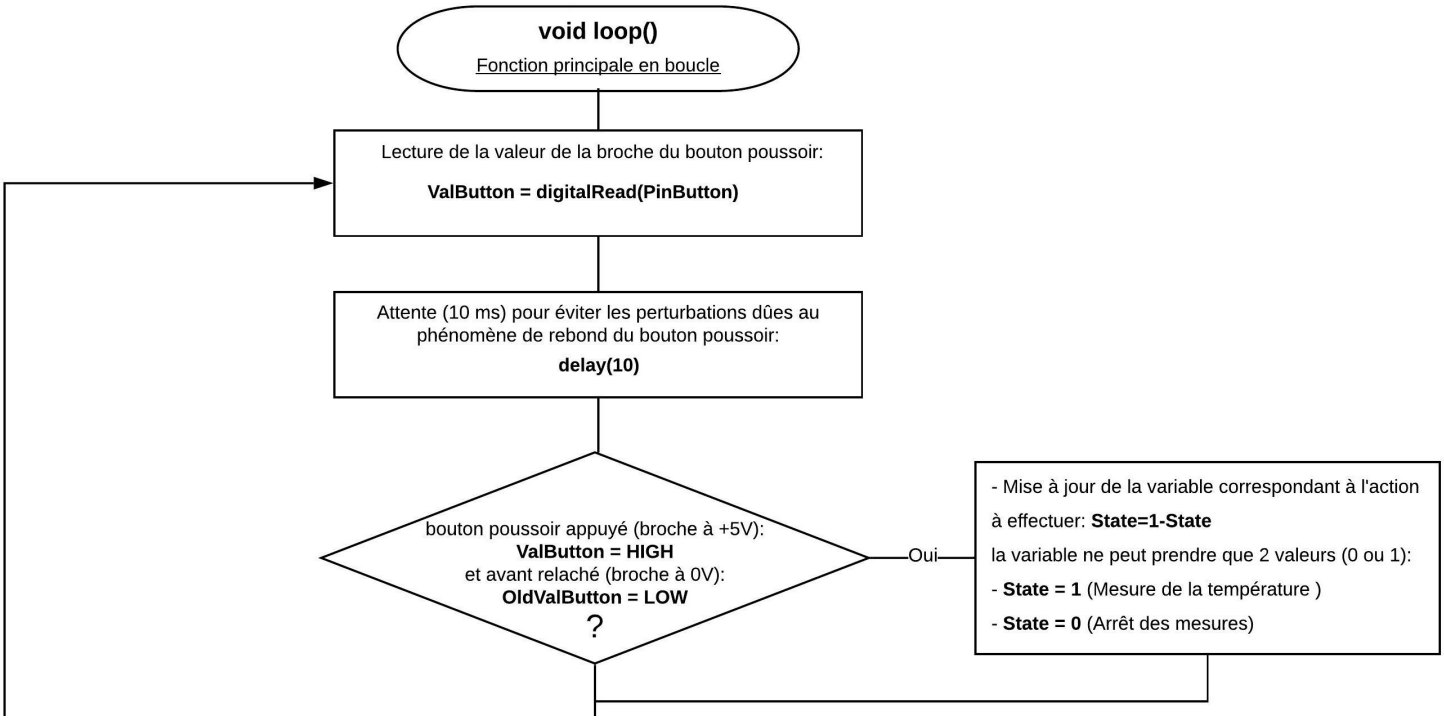
- Déclaration des constantes et variables :

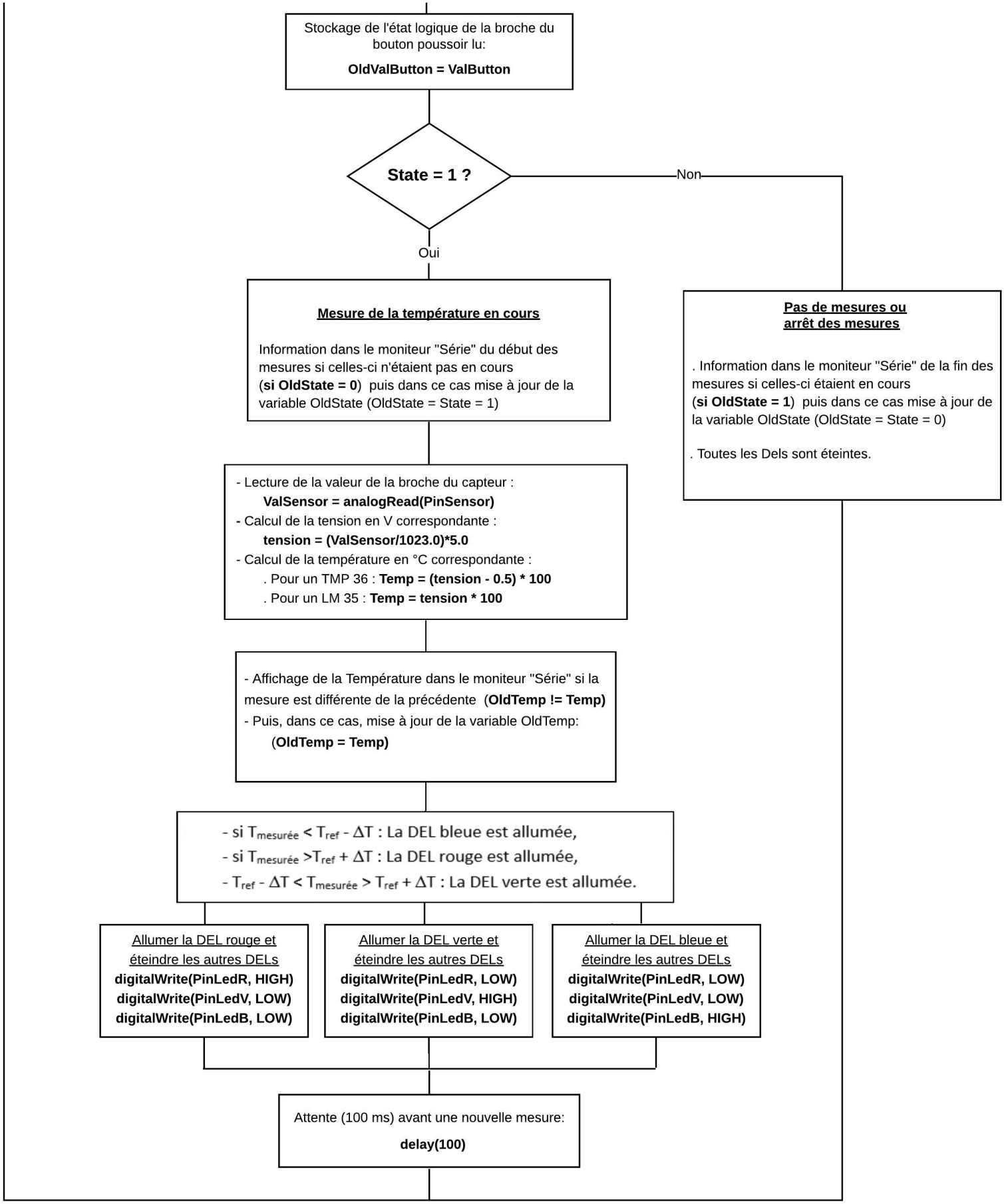
- N° de la broche correspondant au bouton poussoir: **const int PinButton = 12**
- N° de la broche correspondant à la DEL rouge : **const int PinLedR = 8**
- N° de la broche correspondant à la DEL verte : **const int PinLedV = 7**
- N° de la broche correspondant à la DEL bleue : **const int PinLedV = 2**
- N° de la broche correspondant au capteur de température : **const int PinSensor = 0**
- Variable pour stocker la valeur de la broche du bouton poussoir: **int ValButton = 0**
- Variable pour stocker l'ancienne valeur de la broche du bouton poussoir: **int OldValButton = 0**
- Variable correspondant à l'action à effectuer: **int State = 0**
- Variable pour stocker l'ancienne valeur de la variable correspondant à l'action à effectuer: **int OldState = 0**
- Variable pour stocker la valeur de la broche du capteur de température: **int ValSensor = 0**
- Variable pour stocker la valeur en V de la tension correspondante à la valeur de la broche du capteur : **float tension = 0.0**
- Variable correspondant à la température calculée à partir de la valeur de la broche du capteur: **float Temp = 0.0**
- Variable correspondant à la température mesurée précédemment: **float OldTemp = 0.0**
- Variable correspondant à la température de référence: **float TempRef = 20.0**
- Variable correspondant à l'écart de température par rapport à la température de référence: **float DT = 1.0**

- Initialisation des entrées et sorties :

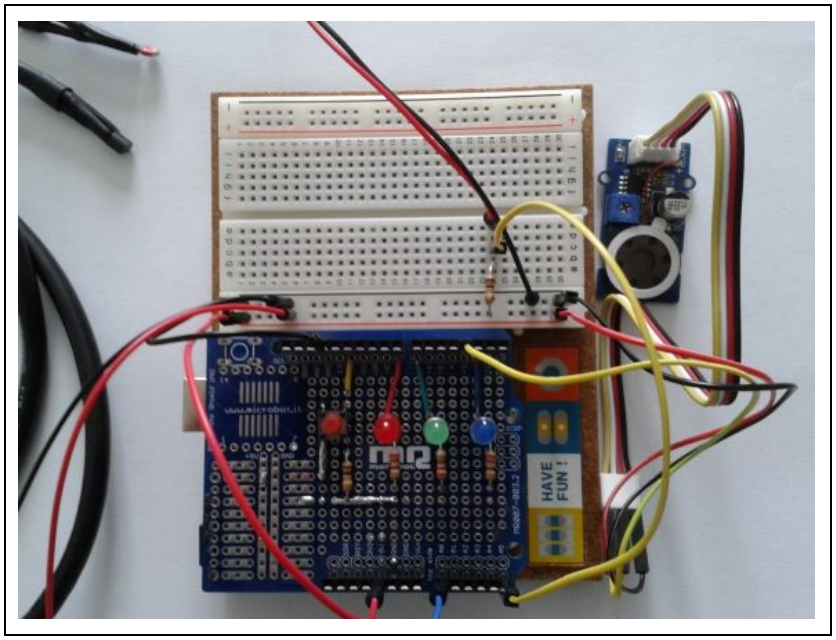
- le débit de communication en nombre de caractères par seconde pour la communication série est fixé à 9600 bauds: **Serial.begin(9600)**
- La broche du bouton poussoir est initialisée comme une entrée digitale. Des données seront donc envoyées depuis cette broche vers le microcontrôleur: **pinMode (PinButton, INPUT)**
- Les broches des DELs rouge, verte et bleue sont initialisées comme des sorties digitales. Des données seront donc envoyées depuis le microcontrôleur vers ces broches.

- Fonction principale en boucle :




Stockage de l'état logique de la broche du bouton poussoir lu:
OldValButton = ValButton
State = 1 ?
Non
Oui
Mesure de la température en cours
Information dans le moniteur "Série" du début des mesures si celles-ci n'étaient pas en cours (si OldState = 0) puis dans ce cas mise à jour de la variable OldState (OldState = State = 1)
Pas de mesures ou arrêt des mesures
. Information dans le moniteur "Série" de la fin des mesures si celles-ci étaient en cours (si OldState = 1) puis dans ce cas mise à jour de la variable OldState (OldState = State = 0)
. Toutes les Dels sont éteintes.
- Lecture de la valeur de la broche du capteur :
ValSensor = analogRead(PinSensor)
- Calcul de la tension en V correspondante :
tension = (ValSensor/1023.0)*5.0
- Calcul de la température en °C correspondante :
. Pour un TMP 36 : Temp = (tension - 0.5) * 100
. Pour un LM 35 : Temp = tension * 100
- Affichage de la Température dans le moniteur "Série" si la mesure est différente de la précédente (OldTemp != Temp)
- Puis, dans ce cas, mise à jour de la variable OldTemp:
(OldTemp = Temp)
- si Tmesurée < Tref - ΔT : La DEL bleue est allumée,
- si Tmesurée >Tref + ΔT : La DEL rouge est allumée,
- Tref - ΔT < Tmesurée > Tref + ΔT : La DEL verte est allumée.
Allumer la DEL rouge et éteindre les autres DELs
digitalWrite(PinLedR, HIGH)
digitalWrite(PinLedV, LOW)
digitalWrite(PinLedB, LOW)
Allumer la DEL verte et éteindre les autres DELs
digitalWrite(PinLedR, LOW)
digitalWrite(PinLedV, HIGH)
digitalWrite(PinLedB, LOW)
Allumer la DEL bleue et éteindre les autres DELs
digitalWrite(PinLedR, LOW)
digitalWrite(PinLedV, LOW)
digitalWrite(PinLedB, HIGH)
Attente (100 ms) avant une nouvelle mesure:
delay(100)

- Activité 4 : Etalonnage d'une thermistance (CTN)



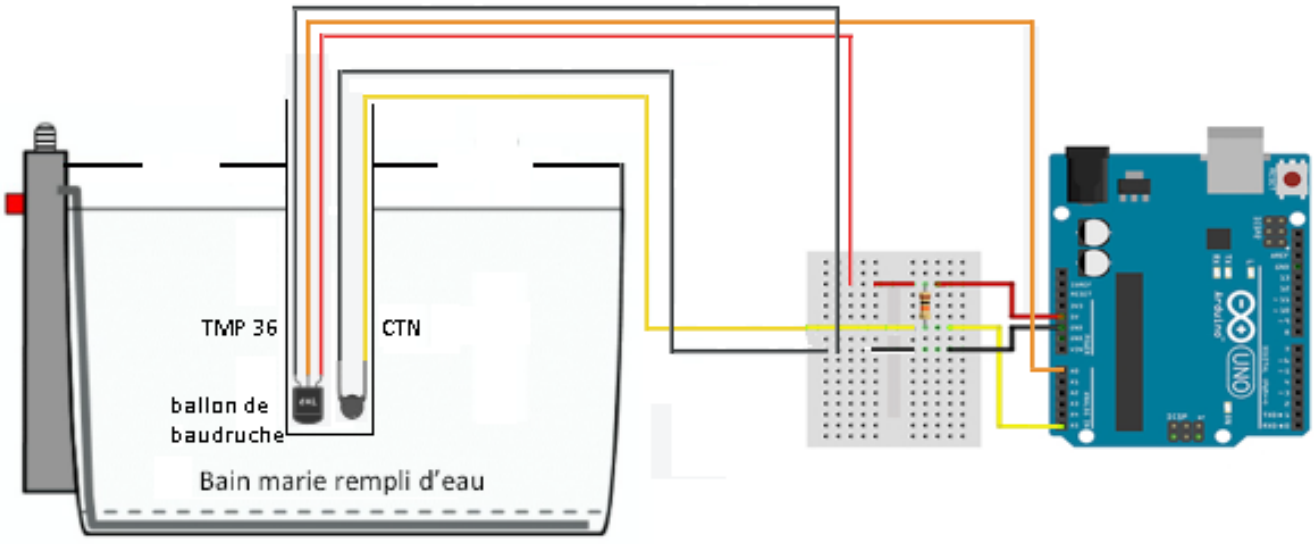
. Objectif

L'objectif de cette activité est d'étalonner une thermistance dont les grandeurs caractéristiques ne sont pas connues, à l'aide d'un capteur de température TMP 36 ou LM 35.

Pour cela, nous allons faire varier la température et on va demander à l'Arduino de mesurer la résistance de la thermistance en fonction de la température qui sera mesurée en parallèle par le capteur.

On pourra alors tracer la caractéristique **résistance = f(température)** qui permettra après de mesurer n'importe quelle température avec cette thermistance grâce à la mesure de sa résistance.

. Le montage

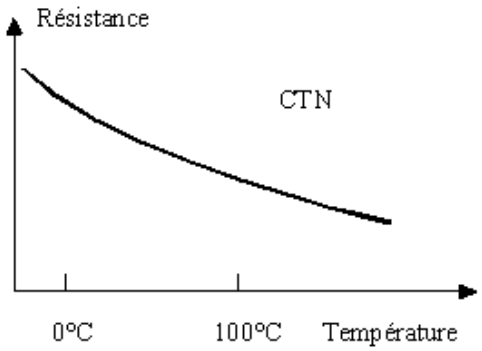


. Les thermistances

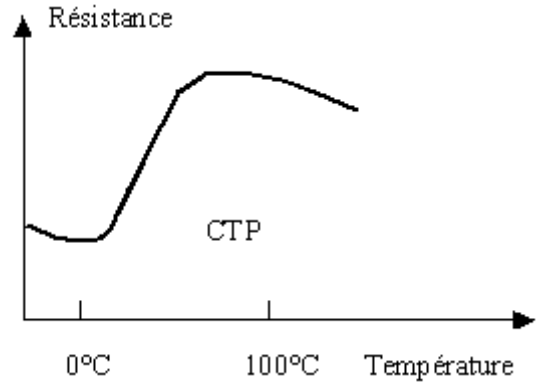
Les thermistances, composées d'un matériau semi-conducteur, combinaison de métaux et de matériaux à base d'oxyde métallique (oxyde de manganèse, cobalt, cuivre et nickel), sont des capteurs de température résistifs, dont la résistance varie de façon importante et non linéaire quand la température change même faiblement.

On distingue deux types de thermistances : les CTN et les CTP.

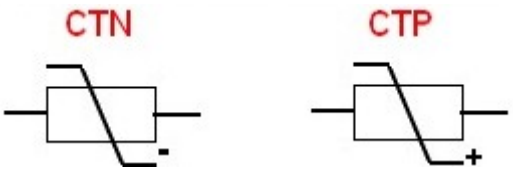
- Les CTN (Coefficient de Température Négatif, en anglais NTC, Negative Temperature Coefficient) sont des thermistances dont la résistance diminue quand la température augmente :



- Les CTP (Coefficient de Température Positif, en anglais PTC, Positive Temperature Coefficient) sont des thermistances, fabriquées à base de titanate de baryum, dont la résistance augmente fortement avec la température dans une plage de température limitée (typiquement entre 0 °C et 100 °C), mais diminue en dehors de cette zone :



Le symbole des thermistances est basé sur celui d'une résistance variable :



Même si la variation de résistance en fonction de la température d'une thermistance n'est pas linéaire, il est tout à fait possible de l'utiliser pour faire des mesures de température avec une grande précision si la loi de variation est connue, notamment avec une CTN.

En effet, Quand l'effet Joule (échauffement dû au passage du courant) est négligeable, on peut exprimer une relation entre la résistance de la CTN et sa température par ***la relation de Steinhart-Hart*** :

$$\frac{1}{T} = A + B\ln(R_T) + C(\ln(R_T))^3$$

- R_T est ***la résistance*** (en ohms) du capteur à la température T (en kelvins).
- A, B et C sont ***les coefficients de Steinhart–Hart*** (donnés par le constructeur ou obtenus expérimentalement avec trois mesures de référence) qui sont des constantes caractéristiques du composant valides à toute température.

Cette formule, valable à toutes les températures, peut être simplifiée sur une plage limitée de températures. La formule devient :

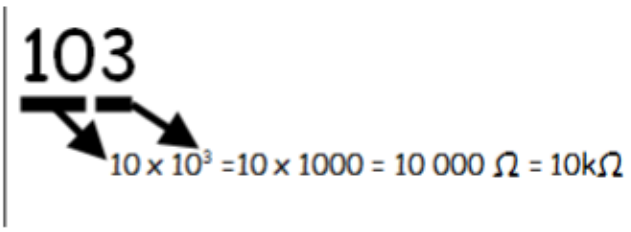
$$\frac{R_T}{R_0} = \exp\left(\beta \times \left(\frac{1}{T} - \frac{1}{T_0}\right)\right)$$

- R_T est ***la résistance*** (en ohms) du capteur à la température T (en kelvins).
- R_0 est la résistance annoncée à une température de référence T_0 (souvent 25 °C).
- β (en kelvins) est un coefficient considéré constant par approximation dont l'usage est limité à une plage de température [T1;T2] :

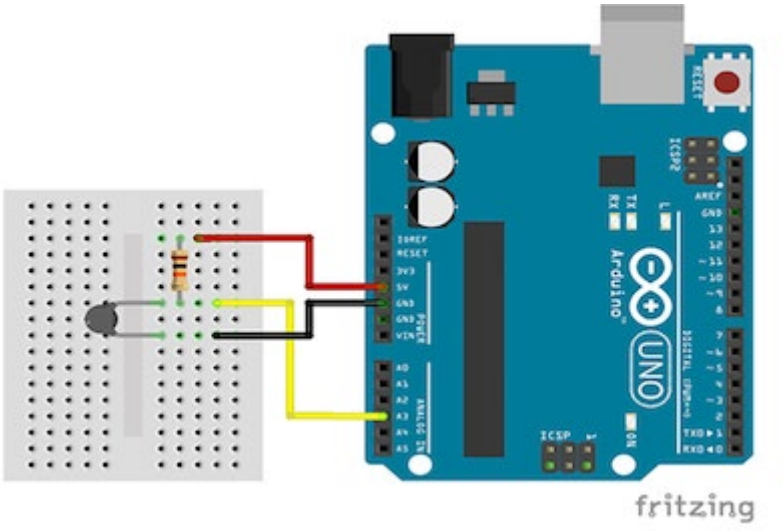
$$\beta = \frac{T_1.T_2}{T_2 - T_1} \times \ln\left(\frac{R_1}{R_2}\right)$$

La principale caractéristique d'une CTN ou d'une CTP est la valeur de la résistance $\mathbf{R_0}$ (en Ω) à la température de 25°C.

Le plus souvent, le marquage du composant indique la valeur de R_0. Il s'agit d'un nombre à 3 chiffres, les 2 premiers chiffres représente la valeur, le 3ème chiffre le coefficient multiplicateur en puissance de 10 :

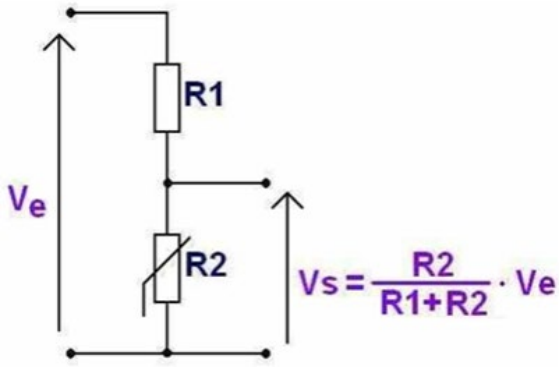


Avec un Arduino, On utilise la thermistance dans un montage (Cf. ci-dessous), avec une résistance fixe de 10 kΩ, qu'on appelle un diviseur de tension.



La thermistance est alimentée en 5V depuis l'Arduino. Le point entre les deux résistances est relié à une broche analogique de l'Arduino et on mesure la tension de cette broche par la fonction **"analogRead (broche) "**.

Diviseur de tension :



En fonction de la température (T), la valeur de la thermistance R2 varie, ce qui modifie le courant qui circule dans le circuit et donc la tension Vs aux bornes de R2.

La résistance R1 de 10kΩ, elle a pour rôle de limiter le courant circulant dans le montage.

La tension Ve est fournie par l'ARDUINO et vaut 5V.

Vs est la tension mesurée par le microcontrôleur. On peut alors calculée R2 :

$$\mathbf{R2 = \frac{Vs\,(R1+R2)}{Ve}} \quad \textbf{donc : R2 (Ve - VS) = R1.Vs}$$

$$\textbf{Soit : R2} = \frac{\mathbf{R1.Vs}}{\mathbf{(Ve - Vs)}}$$

Et tracer la caractéristique **R2 = f(T)**.

. Le programme

Voici le code de l'activité en Python et en langage Arduino.

. Programme en Python (Projet5\Activity4\PY\Activity4.py)

```
# Importations des librairies et définition des fonctions

from PymataExpressDef import *
from ConnectToArduino import *
import time


# Déclaration des constantes et variables

PinTMP = 0
PinCTN = 5
PinButton = 12

ValTMP = 0
ValCTN = 0
TempTMP = 0.0
OldTempTMP = 0.0
Rctn = 0.0
Vctn = 0.0

ValButton = 0
OldValButton = 0
State = 0
OldState = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

Set_AnalogInput_Pin(board, PinTMP)
Set_AnalogInput_Pin(board, PinCTN)
Set_DigitalInput_Pin(board, PinButton)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter.\n")
print("Appuyez sur le bouton poussoir pour commencer les mesures.\n");
```

```
# Boucle principale du programme

while True:
    try:
        ValButton = Digital_Read(board, PinButton)
        time.sleep(0.01)
        if ValButton == 1 and OldValButton == 0:
            State=1-State

        OldValButton = ValButton;

        if State==1:
            if OldState == 0:
               print("\nMesure de la température et de la résistance de la CTN en cours.\n")
               print("Température en degré Celsius ; Résistance CTN en Ohms:")
               OldState=1

            ValTMP = Analog_Read(board, PinTMP)

            # Capteur TMP 36
            TempTMP = ((ValTMP/1023)*5 - 0.5) * 100

            # Capteur LM 35
            # TempTMP = (ValTMP/1023)*5 * 100

            ValCTN = Analog_Read(board, PinCTN)
            Vctn = (ValCTN/1023.0)*5.0
            Rctn = 10000*Vctn/(5-Vctn)

            if OldTempTMP != TempTMP:
                print(round(TempTMP,1), " ; ", round(Rctn,1))
                OldTempTMP = TempTMP

            time.sleep(0.1)

        else:
            if OldState == 1:
                print("\nFin des mesures.\n")
                OldState = 0

    except KeyboardInterrupt:
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module **”ConnectToArduino.py”,** contenant les fonctions de connexion à l’Arduino via le protocole **”Firmata Express”**,

. Le module "**PymataExpressDef.Py**" regroupant toutes les fonctions utiles à l’utilisation de " **Pymata-express** " (fonction de déclaration des entrées et sorties, de lectures, d’écritures…),

. La bibliothèque **"time"** pour la gestion des temps de pause.

- Déclaration des constantes et variables :

. **PinTMP = 0** (cst correspondant au n° de la broche A0 sur laquelle le capteur de température TMP 36 est connecté)

. **PinCTN = 5** (cst correspondant au n° de la broche A5 sur laquelle la thermistance est connectée)

. **ValTMP = 0** (variable pour stocker la valeur de la broche du capteur de température)

. **ValCTN = 0** (variable pour stocker la valeur de la broche de la thermistance)

. **TempTMP = 0** (variable correspondant à la température en °C calculée à partir de la valeur de la broche du capteur de température)

. **OldTempTMP = 0** (variable correspondant à la température en °C calculée précédemment)

. **Vctn = 0** (variable correspondant à la tension calculée en V de la broche de la thermistance)

. **Rctn = 0** (variable correspondant à la valeur de la résistance calculée en Ohms de la thermistance)

. **PinButton = 12** (cst correspondant au n° de la broche sur laquelle le bouton poussoir est connecté)

. **ValButton = 0** (variable pour stocker la valeur de l'état logique de la broche du bouton poussoir)

. **OldValButton = 0** (variable pour stocker la valeur précédente de l'état logique de la broche du bouton poussoir)

. **State=0** (variable correspondant à l'action à effectuer)

. **OldState = 0** (variable pour stocker la valeur précédente de la variable **state**)

. **PortComArduino** (port COM sur lequel l'Arduino est connecté)

- <u>Connexion à l'Arduino</u> :

. Détection du port COM, tentative d'ouverture du port COM sélectionné et connexion à l'Arduino :

PortComArduino = SelectPortCOM()

board = OpenPortCom(PortComArduino)

. Si la connexion à l'Arduino est réussie :

- Déclaration de la broche du capteur de température en entrée analogique :

Set_AnalogInput_Pin(board, PinTMP)

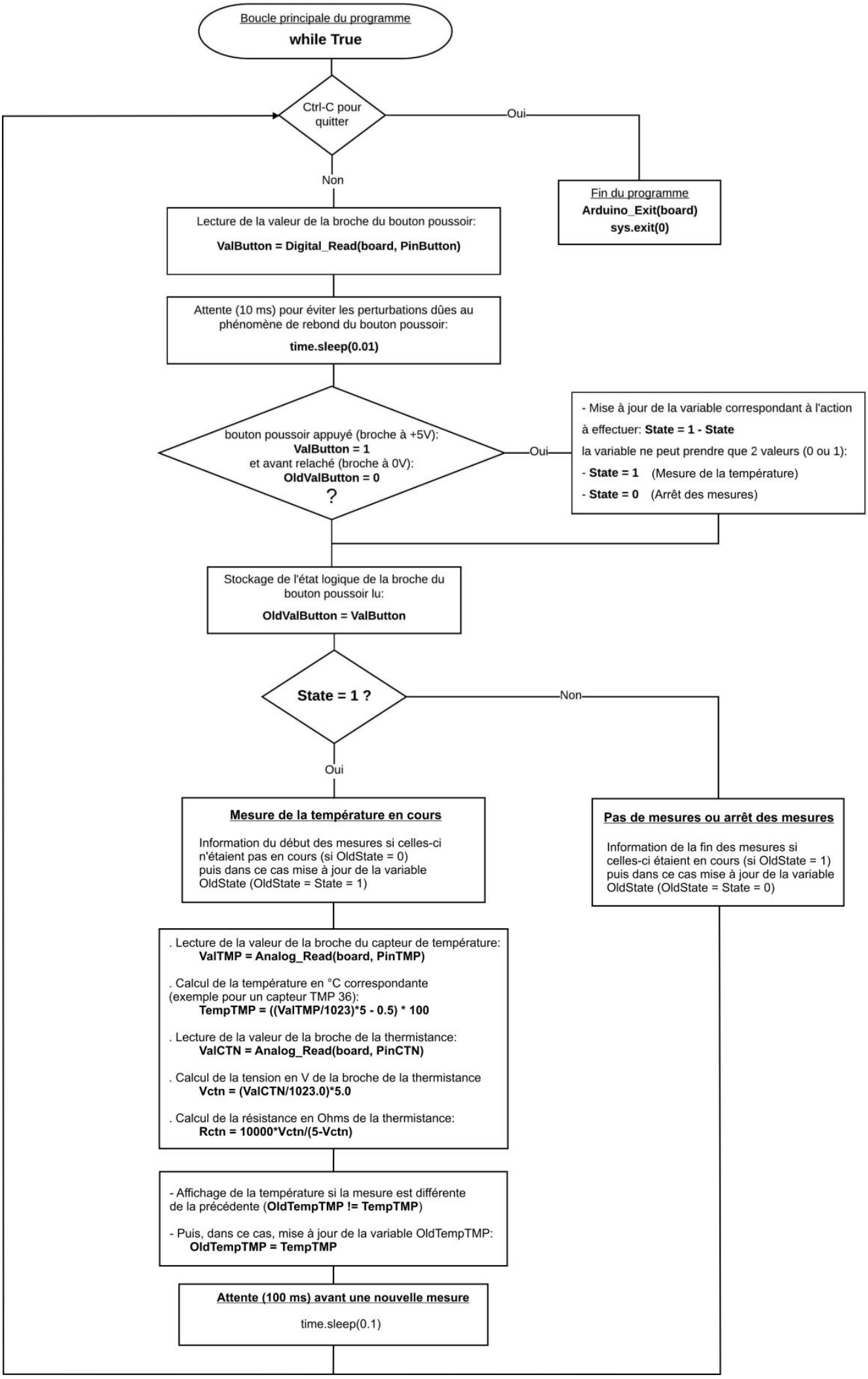
- Déclaration de la broche de la thermistance en entrée analogique :

Set_AnalogInput_Pin(board, PinCTN)

- Déclaration de la broche du bouton poussoir en entrée numérique :

Set_DigitalInput_Pin(board, PinButton)

- <u>Boucle principale du programme (boucle "while True")</u> :


Boucle principale du programme
while True
Ctrl-C pour quitter
Oui
Non
Fin du programme
Arduino_Exit(board)
sys.exit(0)
Lecture de la valeur de la broche du bouton poussoir:
ValButton = Digital_Read(board, PinButton)
Attente (10 ms) pour éviter les perturbations dûes au phénomène de rebond du bouton poussoir:
time.sleep(0.01)
bouton poussoir appuyé (broche à +5V):
ValButton = 1
et avant relaché (broche à 0V):
OldValButton = 0
?
Oui
- Mise à jour de la variable correspondant à l'action à effectuer: State = 1 - State
la variable ne peut prendre que 2 valeurs (0 ou 1):
- State = 1 (Mesure de la température)
- State = 0 (Arrêt des mesures)
Stockage de l'état logique de la broche du bouton poussoir lu:
OldValButton = ValButton
State = 1 ?
Non
Oui
Mesure de la température en cours
Information du début des mesures si celles-ci n'étaient pas en cours (si OldState = 0) puis dans ce cas mise à jour de la variable OldState (OldState = State = 1)
Pas de mesures ou arrêt des mesures
Information de la fin des mesures si celles-ci étaient en cours (si OldState = 1) puis dans ce cas mise à jour de la variable OldState (OldState = State = 0)
. Lecture de la valeur de la broche du capteur de température:
ValTMP = Analog_Read(board, PinTMP)
. Calcul de la température en °C correspondante (exemple pour un capteur TMP 36):
TempTMP = ((ValTMP/1023)*5 - 0.5) * 100
. Lecture de la valeur de la broche de la thermistance:
ValCTN = Analog_Read(board, PinCTN)
. Calcul de la tension en V de la broche de la thermistance
Vctn = (ValCTN/1023.0)*5.0
. Calcul de la résistance en Ohms de la thermistance:
Rctn = 10000*Vctn/(5-Vctn)
- Affichage de la température si la mesure est différente de la précédente (OldTempTMP != TempTMP)
- Puis, dans ce cas, mise à jour de la variable OldTempTMP:
OldTempTMP = TempTMP
Attente (100 ms) avant une nouvelle mesure
time.sleep(0.1)

Résultats dans la fenêtre Python Shell :

```
Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter.

Appuyez sur le bouton poussoir pour commencer les mesures.


Mesure de la température et de la résistance de la CTN en cours.

Température en degré Celsius ; Résistance CTN en Ohms:
17.4  ;  28314.6
17.9  ;  28458.6
17.4  ;  28314.6
17.9  ;  28458.6
17.0  ;  28458.6
17.4  ;  28458.6

Fin des mesures.
```

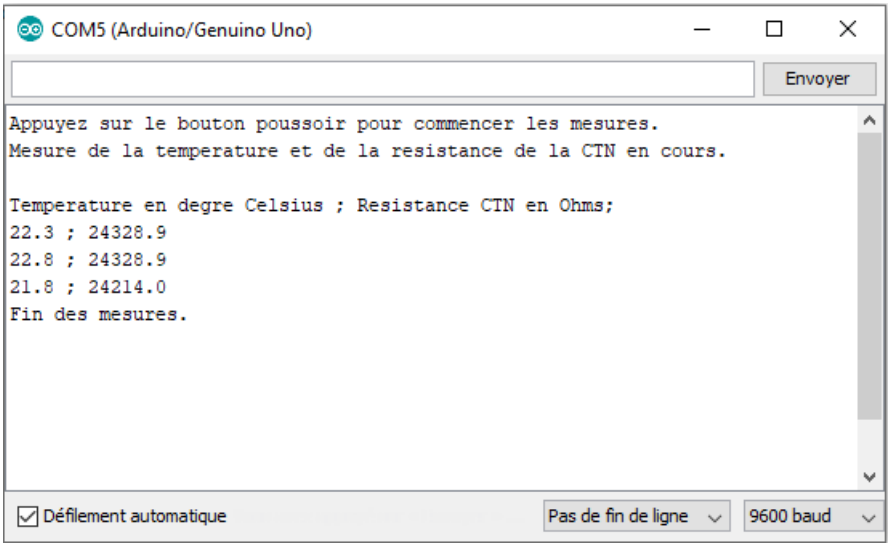
. Programme en langage Arduino (Projet5\Activity4\INO\Activity4.ino)

Activity4

```
// Déclaration des constantes et variables

const int PinTMP = 0;
const int PinCTN = 5;
const int PinButton = 12;


int ValTMP = 0;
int ValCTN = 0;
float TempTMP = 0.0;
float OldTempTMP = 0.0;
float Rctn = 0.0;
float Vctn = 0.0;


int ValButton = 0;
int OldValButton = 0;
int State = 0;
int OldState = 0;


// Initialisation des entrées et sorties

void setup() {
  Serial.begin(9600);
  pinMode(PinButton,INPUT);
  Serial.println("Appuyez sur le bouton poussoir pour commencer les mesures.");
}


// Fonction principale en boucle

void loop() {
  ValButton = digitalRead(PinButton);
  delay(10);
```

```
  if ((ValButton == HIGH)&&(OldValButton == LOW))
  {
  State=1-State;
  }
  OldValButton = ValButton;

  if (State==1)
  {
    if (OldState == 0)
    {
    Serial.println("Mesure de la temperature et de la resistance de la CTN en cours.");
    Serial.println("");
    Serial.println ("Temperature en degre Celsius ; Resistance CTN en Ohms;");
    OldState=1;
    }
    ValTMP = analogRead(PinTMP);
    // Capteur TMP 36
    TempTMP = ((ValTMP/1023.0)*5.0 - 0.5) * 100;
    // Capteur LM 35
    //TempTMP = (ValTMP/1023.0)*5.0 * 100;

    ValCTN = analogRead(PinCTN);
    Vctn = (ValCTN/1023.0)*5.0;
    Rctn = 10000*Vctn/(5-Vctn);

    if (OldTempTMP != TempTMP)
    {
      Serial.print(TempTMP,1);
      Serial.print(" ; ");
      Serial.println(Rctn,1);
      OldTempTMP = TempTMP;
    }

  delay(100);
  }
  else
  {
    if (OldState == 1){
    Serial.println("Fin des mesures.");
    OldState = 0;}
  }
}
```

Résultats dans le moniteur série :



<u>Déroulement du programme</u> :

- <u>Déclaration des constantes et variables</u> :

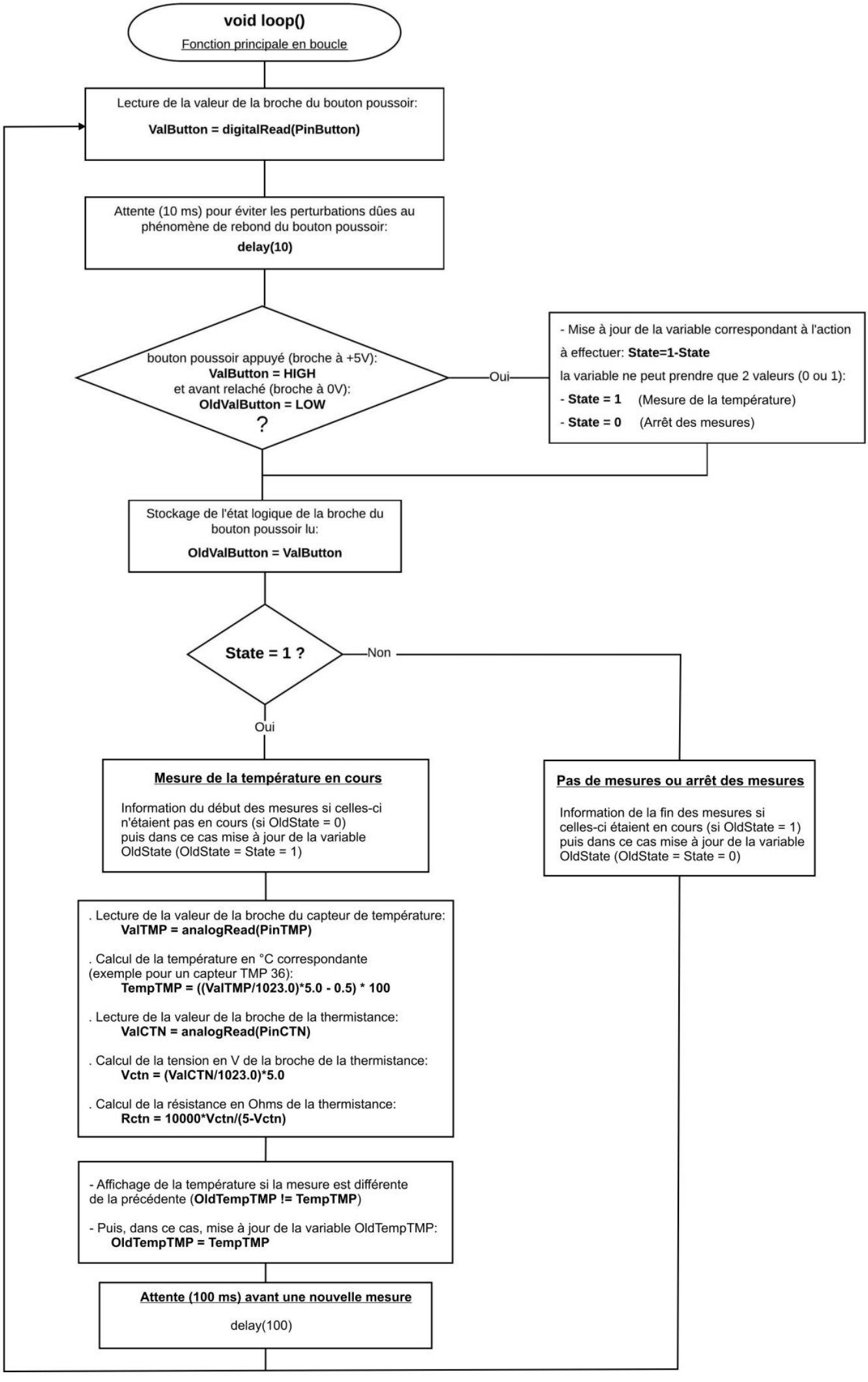
- N° de la broche correspondant au bouton poussoir: **const int PinButton = 12**
- N° de la broche correspondant au capteur de température : **const int PinTMP = 0**
- N° de la broche correspondant à la thermistance: **const int PinCTN = 5**
- Variable pour stocker la valeur de la broche du bouton poussoir: **int ValButton = 0**
- Variable pour stocker l'ancienne valeur de la broche du bouton poussoir: **int OldValButton = 0**
- Variable correspondant à l'action à effectuer: **int State = 0**
- Variable pour stocker l'ancienne valeur de la variable correspondant à l'action à effectuer: **int OldState = 0**
- Variable pour stocker la valeur de la broche du capteur de température: **int ValTMP = 0**
- Variable pour stocker la valeur de la broche de la thermistance: **int ValCTN = 0**
- Variable correspondant à la température calculée à partir de la valeur de la broche du capteur: **float TempTMP = 0.0**
- Variable correspondant à la température mesurée précédemment: **float OldTempTMP = 0.0**
- Variable correspondant à la tension calculée en V de la broche de la thermistance: **float Vctn = 0.0**
- Variable correspondant à la valeur de la résistance calculée en Ohms de la thermistance: **float Rctn = 0.0**

- <u>Initialisation des entrées et sorties</u> :

- Le débit de communication en nombre de caractères par seconde pour la communication série est fixé à 9600 bauds:
  Serial.begin(9600)

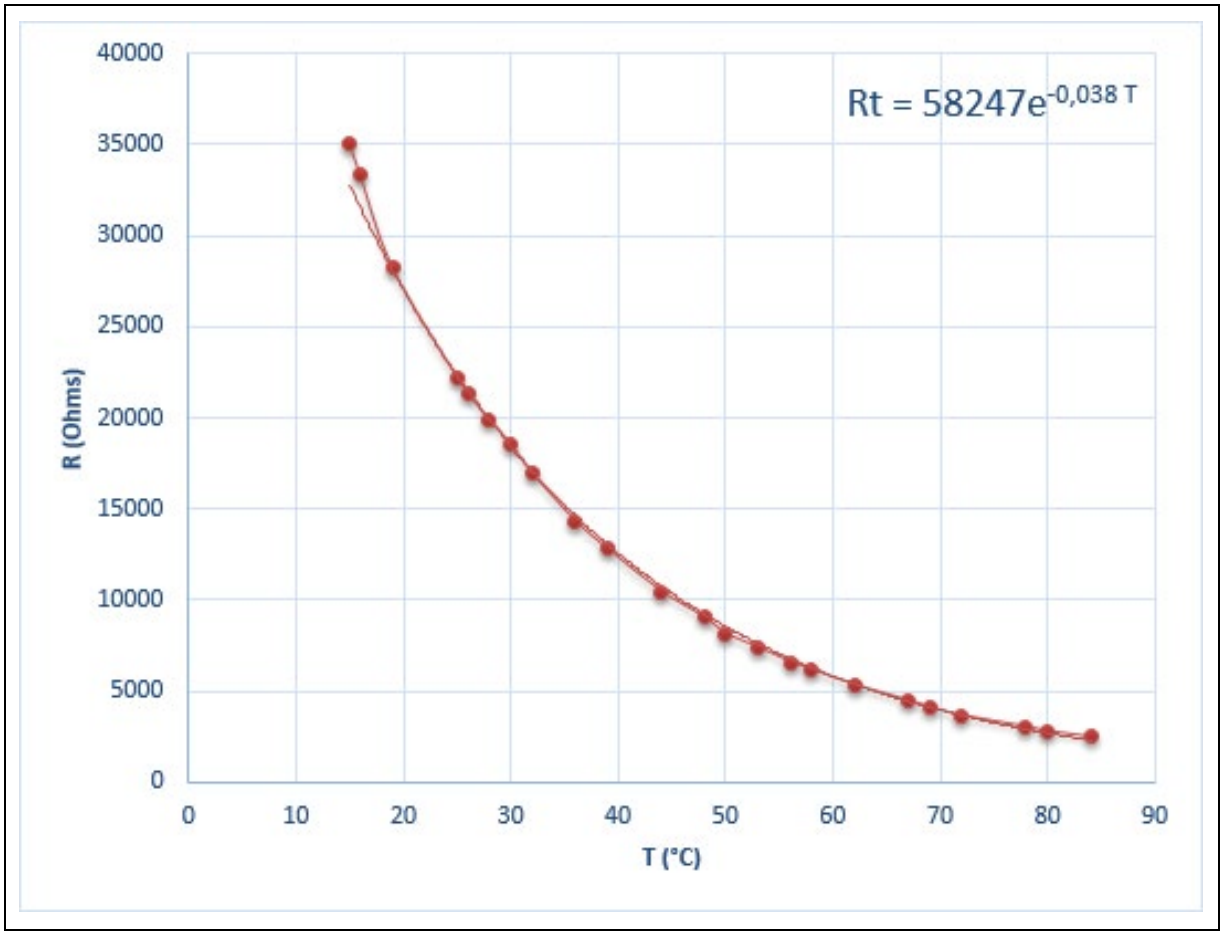
- La broche du bouton poussoir est initialisée comme une entrée digitale. Des données seront donc envoyées depuis cette broche vers le microcontrôleur:
  pinMode (PinButton, INPUT)

- <u>Fonction principale en boucle</u> :


void loop()
Fonction principale en boucle
Lecture de la valeur de la broche du bouton poussoir:
ValButton = digitalRead(PinButton)
Attente (10 ms) pour éviter les perturbations dûes au phénomène de rebond du bouton poussoir:
delay(10)
bouton poussoir appuyé (broche à +5V):
ValButton = HIGH
et avant relaché (broche à 0V):
OldValButton = LOW
?
Oui
- Mise à jour de la variable correspondant à l'action à effectuer: State=1-State
la variable ne peut prendre que 2 valeurs (0 ou 1):
- State = 1 (Mesure de la température)
- State = 0 (Arrêt des mesures)
Stockage de l'état logique de la broche du bouton poussoir lu:
OldValButton = ValButton
State = 1 ?
Non
Oui
Mesure de la température en cours
Information du début des mesures si celles-ci n'étaient pas en cours (si OldState = 0) puis dans ce cas mise à jour de la variable OldState (OldState = State = 1)
Pas de mesures ou arrêt des mesures
Information de la fin des mesures si celles-ci étaient en cours (si OldState = 1) puis dans ce cas mise à jour de la variable OldState (OldState = State = 0)
. Lecture de la valeur de la broche du capteur de température:
ValTMP = analogRead(PinTMP)
. Calcul de la température en °C correspondante (exemple pour un capteur TMP 36):
TempTMP = ((ValTMP/1023.0)*5.0 - 0.5) * 100
. Lecture de la valeur de la broche de la thermistance:
ValCTN = analogRead(PinCTN)
. Calcul de la tension en V de la broche de la thermistance:
Vctn = (ValCTN/1023.0)*5.0
. Calcul de la résistance en Ohms de la thermistance:
Rctn = 10000*Vctn/(5-Vctn)
- Affichage de la température si la mesure est différente de la précédente (OldTempTMP != TempTMP)
- Puis, dans ce cas, mise à jour de la variable OldTempTMP:
OldTempTMP = TempTMP
Attente (100 ms) avant une nouvelle mesure
delay(100)

. Exploitation des mesures

En chauffant progressivement le bain-marie, on relève la température (T) donnée par le capteur TMP 36 et la résistance (Rt) de la CTN. Puis on trace la caractéristique **Rt = f(T)** :



La modélisation donne : $\mathbf{Rt} = \mathbf{5,82.10^4}\, e^{(-\frac{T}{26,32})}$ (T en °C, R en Ω).

On en déduit alors l'expression de la température en fonction de la résistance de la CTN, dans la plage de températures de l'expérience :

$$e^{(-\frac{T}{26,32})} = \frac{Rt}{5,82.10^4}$$

$$-\frac{T}{26,32} = ln(\frac{Rt}{5,82.10^4})$$

$$T = 26,32\ ln(\frac{5,82.10^4}{Rt})$$

Ainsi, avec l'Arduino, pour n'importe quelle température dans la plage de l'expérience, par mesure de la résistance de la CTN, on pourra déterminer la température correspondante.

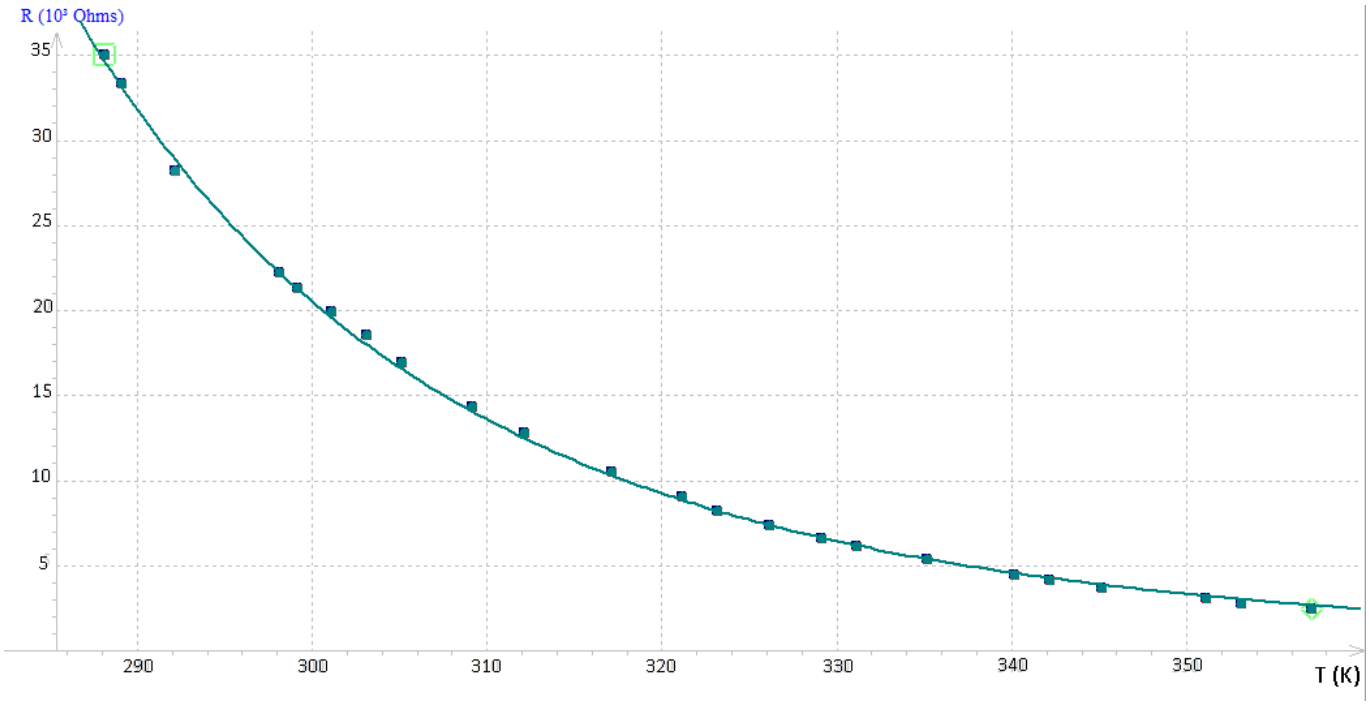
On peut également modéliser la caractéristique Rt = f(T) de notre CTN avec la relation simplifiée de ***Steinhart-Hart*** (T en K):

$$\frac{R_T}{R_0} = \exp\left(\beta \times \left(\frac{1}{T} - \frac{1}{T_0}\right)\right)$$

A partir du graphe, on détermine une des deux grandeurs caractéristiques de la CTN:

→ à T_0 = 298,15 K, R_0 = 22,2 kΩ

La modélisation suivant cette relation à l'aide du logiciel Régressi donne la valeur de β (en K) :



Expression du modèle

R(T)=22200*exp(b*((1/T)-(1/298.15)))

Résultats de la modélisation

Ecart expérience-modèle
1,7 % sur R(T)
Ecart quad. R=280,9 Ohms

b=(3,82 ±0,06) 10³

Soit : **β = 3820 K**

Dans la plage de températures de l'expérience, la relation simplifiée de ***Steinhart-Hart*** est vérifiée.

On en déduit alors l'expression de la température (en K) en fonction de la résistance de la CTN (en Ohms) :

$R_T = R_0\, e^{\beta\left(\frac{1}{T}-\frac{1}{To}\right)}$ donc : $ln(R_T) = ln(R_0\, e^{\beta\left(\frac{1}{T}-\frac{1}{To}\right)})$

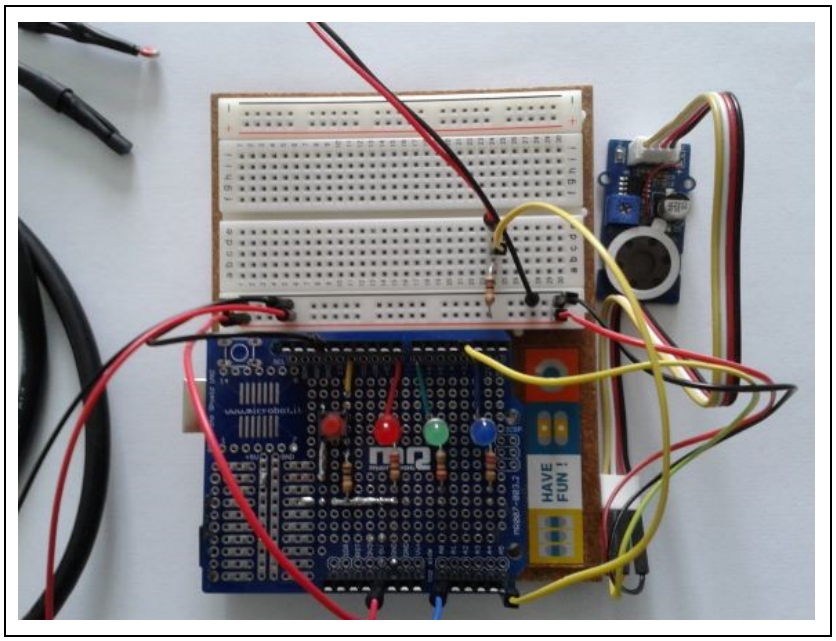
$ln(R_T)$ = $ln(R_0) + \beta\left(\frac{1}{T}-\frac{1}{To}\right)$

$\beta\left(\frac{1}{T}-\frac{1}{To}\right)$ = $ln(R_T)$ - $ln(R_0)$ = $ln(\frac{R_T}{R_0})$

Soit :

$$\frac{1}{T} = \frac{1}{\beta}\, ln\left(\frac{R_T}{R_0}\right) + \frac{1}{To}$$

- Activité 5 : Mesure de températures avec une thermistance CTN



. Objectif

Pour mesurer une température avec une thermistance CTN, il faut connaitre ses grandeurs caractéristiques. Le plus souvent, le constructeur fourni les valeurs suivantes :

- La valeur de sa résistance R_0 (résistance nominale en Ω) à la température de référence T_0 = 25 °C (298,15 K)

- La valeur de β (en K)

- La plage de température pour laquelle la relation entre la température T (en K) et R_T, la résistance (en ohms) de la CTN à cette température, est vérifiée :

$$\frac{1}{T} = \frac{1}{\beta}\, ln\left(\frac{R_T}{R_0}\right) + \frac{1}{To}$$

On en déduit :

$$T\ (en\ °C) = \frac{1}{\frac{1}{\beta}\, ln\left(\frac{R_T}{R_0}\right) + \frac{1}{To}} - 273{,}15$$

En l'absence de ces grandeurs caractéristiques, il faut procéder à un étalonnage (cf. activité précédente), afin de les déterminer expérimentalement.

L'objectif de l'activité est d'écrire un programme généraliste permettant de mesurer une température avec une thermistance CTN quelconque dont les grandeurs caractéristiques sont connues.

. Le programme

Le code de l'activité en Python ou en langage Arduino demande, à l'initialisation du programme, de renseigner les valeurs de T_0, R_0 et β afin de pouvoir calculer la température à partir de la mesure de la résistance de la CTN.

. Programme en Python (Projet5\Activity5\PY\Activity5.py)

```
# Importations des librairies et définition des fonctions

from PymataExpressDef import *
from ConnectToArduino import *
import time
import math

def InputVal(msg, valmin, valmax, unit):
    InputValid = False
    while InputValid==False:
        val = input(msg+"(valeur entre {0} et {1} {2}) :" .format(valmin, valmax, unit))
        try:
            val = int(val)
            assert val >= valmin and val <= valmax

        except AssertionError:
            print("La valeur saisie n'est pas entre {0} et {1} {2} !".format(valmin, valmax, unit))

        except:
            print("vous n'avez pas saisi une valeur entre {0} et {1} {2} °C !".format(valmin, valmax, unit))

        else:
            InputValid = True

    return val


# Déclaration des constantes et variables

PinCTN = 5
PinButton = 12

ValCTN = 0
Temp = 0.0
OldTemp = 0.0
Rt = 0.0
Vctn = 0.0
TempRef = 0
Ro = 0
B = 0

ValButton = 0
OldValButton = 0
State = 0
OldState = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

Set_AnalogInput_Pin(board, PinCTN)
Set_DigitalInput_Pin(board, PinButton)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter.\n")

TempRef = InputVal("Veuillez saisir la température de référence ",1,100,"°C")
Ro = InputVal("Veuillez saisir la valeur de la resistance nominale ",1,200000,"Ohms")
B = InputVal("Veuillez saisir la constante B ",1,10000,"K")

print("\nAppuyez sur le bouton poussoir pour commencer les mesures.\n");
```

```
# Boucle principale du programme

while True:
    try:
        ValButton = Digital_Read(board, PinButton)
        time.sleep(0.01)
        if ValButton == 1 and OldValButton == 0:
            State=1-State

        OldValButton = ValButton;

        if State==1:
            if OldState == 0:
               print("\nMesure de la température en cours.\n")
               print("Résistance CTN en Ohms ; Température en degré Celsius:")
               OldState=1

            ValCTN = Analog_Read(board, PinCTN)
            Vctn = (ValCTN/1023.0)*5.0
            Rt = 10000*Vctn/(5-Vctn)
            Temp = 1.0/(math.log(Rt/Ro)/B+1/(TempRef+273.15))-273.15

            if OldTemp != Temp:
                print(round(Rt,1), " ; ", round(Temp,1))
                OldTemp = Temp

            time.sleep(0.1)

        else:
            if OldState == 1:
                print("\nFin des mesures.\n")
                OldState = 0

    except KeyboardInterrupt:
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module **"ConnectToArduino.py",** contenant les fonctions de connexion à l'Arduino via le protocole **"Firmata Express"**,

. Le module "**PymataExpressDef.Py**" regroupant toutes les fonctions utiles à l'utilisation de " **Pymata-express** " (fonction de déclaration des entrées et sorties, de lectures, d'écritures…),

. La bibliothèque **"time"** pour la gestion des temps de pause,

. La bibliothèque **"math"** pour les calculs de température,

. La fonction **"InputVal"** pour saisir les grandeurs caractéristiques de la thermistance.

- Déclaration des constantes et variables :

. **PinCTN = 5** (cst correspondant au n° de la broche A5 sur laquelle la thermistance est connectée)

. **ValCTN = 0** (variable pour stocker la valeur de la broche de la thermistance)

. **Temp = 0** (variable correspondant à la température en °C calculée à partir de la valeur de la broche de la thermistance)

. **OldTemp = 0** (variable correspondant à la température en °C calculée précédemment)

. **Vctn = 0** (variable correspondant à la tension calculée en V de la broche de la thermistance)

. **Rt = 0** (variable correspondant à la valeur de la résistance calculée en Ohms de la thermistance)

. **TempRef = 0** (variable correspondant à la température de référence de la thermistance)

. **Ro = 0** (variable correspondant à la résistance de la thermistance à la température de référence)

. **B = 0** (variable correspondant à la grandeur caractéristique β de la thermistance)

. **PinButton = 12** (cst correspondant au n° de la broche sur laquelle le bouton poussoir est connecté)

. **ValButton = 0** (variable pour stocker la valeur de l'état logique de la broche du bouton poussoir)

. **OldValButton = 0** (variable pour stocker la valeur précédente de l'état logique de la broche du bouton poussoir)

. **State=0** (variable correspondant à l'action à effectuer)

. **OldState = 0** (variable pour stocker la valeur précédente de la variable **state**)

. **PortComArduino** (port COM sur lequel l'Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM, tentative d'ouverture du port COM sélectionné et connexion à l'Arduino :

PortComArduino = SelectPortCOM()

board = OpenPortCom(PortComArduino)

. Si la connexion à l'Arduino est réussie :

- Déclaration de la broche de la thermistance en entrée analogique :

Set_AnalogInput_Pin(board, PinCTN)

- Déclaration de la broche du bouton poussoir en entrée numérique :

Set_DigitalInput_Pin(board, PinButton)

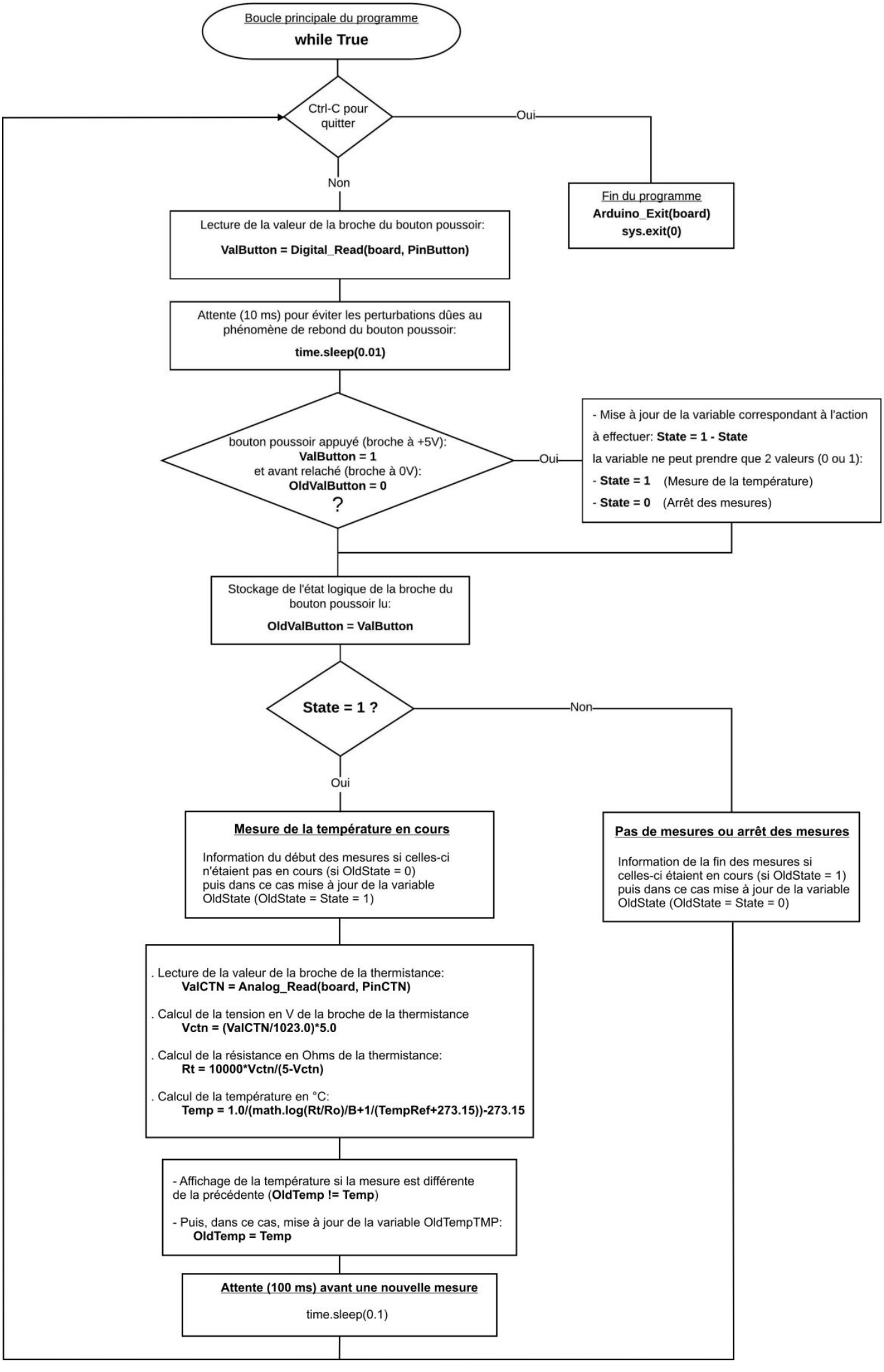
- Saisie des grandeurs caractéristiques de la thermistance :

TempRef = InputVal("Veuillez saisir la température de référence ",1,100,"°C")

Ro = InputVal("Veuillez saisir la valeur de la résistance nominale ",1,200000,"Ohms")

B = InputVal("Veuillez saisir la constante B ",1,10000,"K")

- Boucle principale du programme (boucle "while True") :


Boucle principale du programme
while True
Ctrl-C pour quitter
Oui
Non
Fin du programme
Arduino_Exit(board)
sys.exit(0)
Lecture de la valeur de la broche du bouton poussoir:
ValButton = Digital_Read(board, PinButton)
Attente (10 ms) pour éviter les perturbations dûes au phénomène de rebond du bouton poussoir:
time.sleep(0.01)
bouton poussoir appuyé (broche à +5V):
ValButton = 1
et avant relaché (broche à 0V):
OldValButton = 0
?
Oui
- Mise à jour de la variable correspondant à l'action à effectuer: State = 1 - State
la variable ne peut prendre que 2 valeurs (0 ou 1):
- State = 1 (Mesure de la température)
- State = 0 (Arrêt des mesures)
Stockage de l'état logique de la broche du bouton poussoir lu:
OldValButton = ValButton
State = 1 ?
Non
Oui
Mesure de la température en cours
Information du début des mesures si celles-ci n'étaient pas en cours (si OldState = 0) puis dans ce cas mise à jour de la variable OldState (OldState = State = 1)
Pas de mesures ou arrêt des mesures
Information de la fin des mesures si celles-ci étaient en cours (si OldState = 1) puis dans ce cas mise à jour de la variable OldState (OldState = State = 0)
. Lecture de la valeur de la broche de la thermistance:
ValCTN = Analog_Read(board, PinCTN)
. Calcul de la tension en V de la broche de la thermistance
Vctn = (ValCTN/1023.0)*5.0
. Calcul de la résistance en Ohms de la thermistance:
Rt = 10000*Vctn/(5-Vctn)
. Calcul de la température en °C:
Temp = 1.0/(math.log(Rt/Ro)/B+1/(TempRef+273.15))-273.15
- Affichage de la température si la mesure est différente de la précédente (OldTemp != Temp)
- Puis, dans ce cas, mise à jour de la variable OldTempTMP:
OldTemp = Temp
Attente (100 ms) avant une nouvelle mesure
time.sleep(0.1)

Résultats dans la fenêtre Python Shell :

```
Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter.

Veuillez saisir la température de référence (valeur entre 1 et 100 °C) :25
Veuillez saisir la valeur de la résistance nominale (valeur entre 1 et 200000 Ohms) :22200
Veuillez saisir la constante B (valeur entre 1 et 10000 K) :3820

Appuyez sur le bouton poussoir pour commencer les mesures.


Mesure de la température en cours.

Résistance CTN en Ohms ; Température en degré Celsius:
28314.6  ;  19.4
28171.6  ;  19.6
28314.6  ;  19.4
28458.6  ;  19.3
28314.6  ;  19.4
28458.6  ;  19.3
28314.6  ;  19.4
28458.6  ;  19.3

Fin des mesures.

>>>
```

. Programme en langage Arduino (Projet5\Activity5\INO\Activity5.ino)

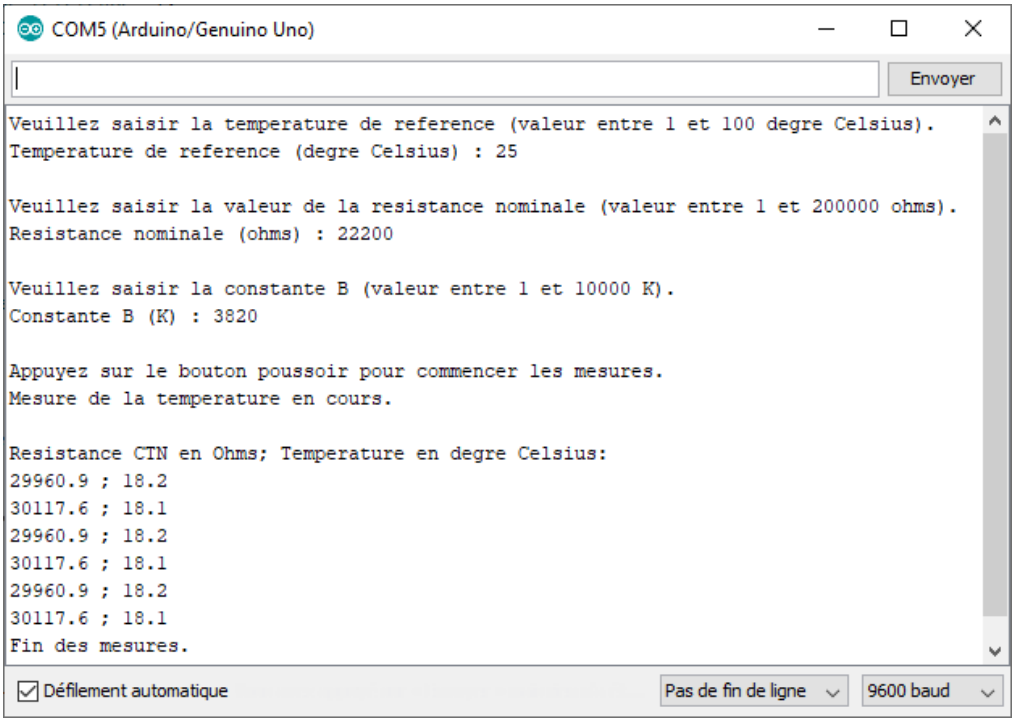
Activity5

```
// Inclusion des librairies

#include <math.h>

// Déclaration des constantes et variables

const int PinCTN = 5;
const int PinButton = 12;

int ValCTN = 0;
float Temp = 0.0;
float OldTemp = 0.0;
float Rt = 0.0;
float Vctn = 0.0;
int TempRef = 0;
long Ro = 0;
int B = 0;

int ValButton = 0;
int OldValButton = 0;
int State = 0;
int OldState = 0;


// Initialisation des entrées et sorties

void setup() {
  Serial.begin(9600);
  pinMode(PinButton,INPUT);
```

```
  Serial.print("Veuillez saisir la temperature de reference ");
  Serial.println("(valeur entre 1 et 100 degre Celsius).");
   while(TempRef<1 || TempRef>100)
  {
    TempRef=Serial.parseInt();
  }
  Serial.print("Temperature de reference (degre Celsius) : ");
  Serial.println(TempRef);
  Serial.println("");

  Serial.print("Veuillez saisir la valeur de la resistance nominale ");
  Serial.println("(valeur entre 1 et 200000 ohms).");
   while(Ro<1 || Ro>200000)
  {
    Ro=Serial.parseInt();
  }
  Serial.print("Resistance nominale (ohms) : "); Serial.println(Ro);
  Serial.println("");

  Serial.print("Veuillez saisir la constante B ");
  Serial.println("(valeur entre 1 et 10000 K).");
   while(B<1 || B>10000)
  {
    B=Serial.parseInt();
  }
  Serial.print("Constante B (K) : "); Serial.println(B);
  Serial.println("");

  Serial.println("Appuyez sur le bouton poussoir pour commencer les mesures.");
}


// Fonction principale en boucle

void loop() {
  ValButton = digitalRead(PinButton);
  delay(10);

  if ((ValButton == HIGH)&&(OldValButton == LOW))
  {
  State=1-State;
  }
  OldValButton = ValButton;
```

```
  if (State==1)
  {
    if (OldState == 0)
    {
    Serial.println("Mesure de la temperature en cours.");
    Serial.println("");
    Serial.println ("Resistance CTN en Ohms; Temperature en degre Celsius:");
    OldState=1;
    }

    ValCTN = analogRead(PinCTN);
    Vctn = (ValCTN/1023.0)*5.0;
    Rt = 10000*Vctn/(5-Vctn);
    Temp = 1.0/(log(Rt/Ro)/B+1/(TempRef+273.15))-273.15;

    if (OldTemp != Temp)
    {
      Serial.print(Rt,1);
      Serial.print(" ; ");
      Serial.println(Temp,1);
      OldTemp = Temp;
    }

  delay(100);
  }
  else
  {
    if (OldState == 1){
    Serial.println("Fin des mesures.");
    OldState = 0;}
  }
}
```

Résultats dans le moniteur série :



Déroulement du programme :

- Inclusion des librairies :

Pour effectuer les calculs de température, le code nécessite l'importation de la librairie "**math.h**".

- Déclaration des constantes et variables :

- N° de la broche correspondant au bouton poussoir: **const int PinButton = 12**
- N° de la broche correspondant à la thermistance: **const int PinCTN = 5**
- Variable pour stocker la valeur de la broche du bouton poussoir: **int ValButton = 0**
- Variable pour stocker l'ancienne valeur de la broche du bouton poussoir: **int OldValButton = 0**
- Variable correspondant à l'action à effectuer: **int State = 0**
- Variable pour stocker l'ancienne valeur de la variable correspondant à l'action à effectuer: **int OldState = 0**
- Variable pour stocker la valeur de la broche de la thermistance: **int ValCTN = 0**
- Variable correspondant à la température calculée à partir de la valeur de la broche de la thermistance: **float Temp = 0**
- Variable correspondant à la température mesurée précédemment: **float OldTemp = 0**
- Variable correspondant à la tension calculée en V de la broche de la thermistance: **float Vctn = 0**
- Variable correspondant à la valeur de la résistance calculée en Ohms de la thermistance: **float Rt = 0**
- Variable correspondant à la température de référence de la thermistance: **int TempRef = 0**
- Variable correspondant à la résistance de la thermistance à la température de référence: **long Ro = 0**
- Variable correspondant à la grandeur caractéristique β de la thermistance**: int B = 0**

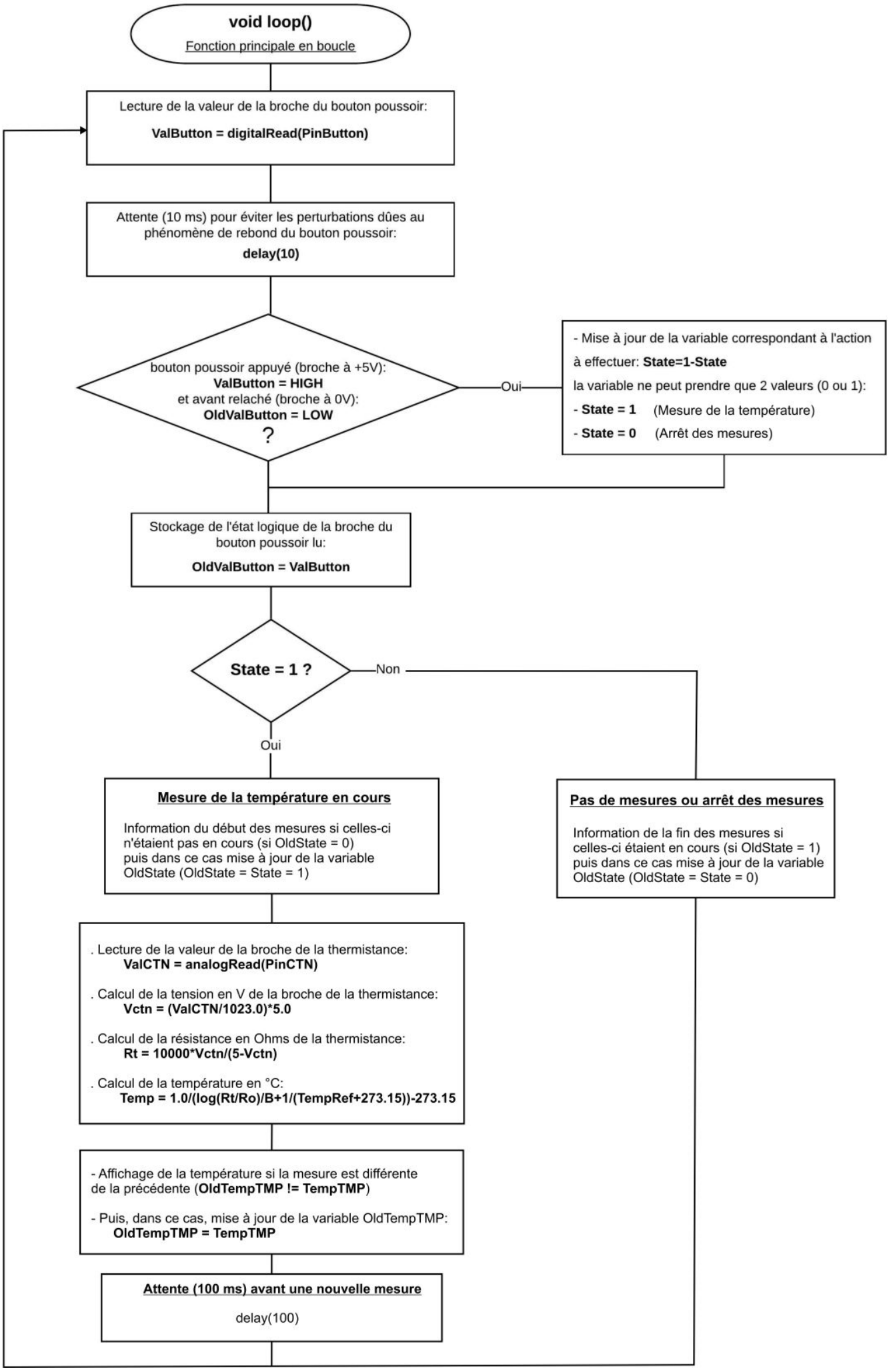
- Initialisation des entrées et sorties :

- Le débit de communication en nombre de caractères par seconde pour la communication série est fixé à 9600 bauds:
  Serial.begin(9600)

- La broche du bouton poussoir est initialisée comme une entrée digitale. Des données seront donc envoyées depuis cette broche vers le microcontrôleur:
  pinMode (PinButton, INPUT)

- Saisie des valeurs de $\mathbf{T_0}$, $\mathbf{R_0}$ et **β** afin de pouvoir calculer la température à partir de la mesure de la résistance de la CTN.

- Fonction principale en boucle :


void loop()
Fonction principale en boucle
Lecture de la valeur de la broche du bouton poussoir:
ValButton = digitalRead(PinButton)
Attente (10 ms) pour éviter les perturbations dûes au phénomène de rebond du bouton poussoir:
delay(10)
bouton poussoir appuyé (broche à +5V):
ValButton = HIGH
et avant relaché (broche à 0V):
OldValButton = LOW
?
Oui
- Mise à jour de la variable correspondant à l'action à effectuer: State=1-State
la variable ne peut prendre que 2 valeurs (0 ou 1):
- State = 1 (Mesure de la température)
- State = 0 (Arrêt des mesures)
Stockage de l'état logique de la broche du bouton poussoir lu:
OldValButton = ValButton
State = 1 ?
Non
Oui
Mesure de la température en cours
Information du début des mesures si celles-ci n'étaient pas en cours (si OldState = 0) puis dans ce cas mise à jour de la variable OldState (OldState = State = 1)
Pas de mesures ou arrêt des mesures
Information de la fin des mesures si celles-ci étaient en cours (si OldState = 1) puis dans ce cas mise à jour de la variable OldState (OldState = State = 0)
. Lecture de la valeur de la broche de la thermistance:
ValCTN = analogRead(PinCTN)
. Calcul de la tension en V de la broche de la thermistance:
Vctn = (ValCTN/1023.0)*5.0
. Calcul de la résistance en Ohms de la thermistance:
Rt = 10000*Vctn/(5-Vctn)
. Calcul de la température en °C:
Temp = 1.0/(log(Rt/Ro)/B+1/(TempRef+273.15))-273.15
- Affichage de la température si la mesure est différente de la précédente (OldTempTMP != TempTMP)
- Puis, dans ce cas, mise à jour de la variable OldTempTMP:
OldTempTMP = TempTMP
Attente (100 ms) avant une nouvelle mesure
delay(100)