Module FISA TI
IOT & Réseaux

Création d'un capteur de température IdO via MODBUS ASCII

Par

Alexis GUILLOTEAU / Didier JUGE-HUBERT

V2023-04

Sommaire

1 Remarques Générales

1.1 Préambule

Pour le TP, vous avez un répertoire « C:\IOT_TP ». Ce répertoire est le répertoire de votre projet.

a) Créer un fichier WORD qui sera le support de votre compte-rendu de mini projet
b) Tous les fichiers doivent obligatoirement être dans ce répertoire
c) Vous aurez à créer des sous-répertoires :
   a. Pour chaque partie (projet Simulide) réalisé avec le logiciel Simulide
   b. Pour la partie Python

REMARQUE IMPORTANTE

Tout document, fichier, ne se trouvant pas dans votre répertoire, pourra être détruit sans aucun préavis.

1.2 Introduction

Objectif : Mettre en œuvre un réseau de type MODBUS ASCII pour permettre la communication entre deux systèmes : un PC et un boîtier intelligent comportant un capteur de température.

Ce boîtier est raccordé à un port USB qui émule un port série RS232C. Il s'agira :

a) D'appréhender le protocole MODBUS ASCII
b) De concevoir un capteur de température autour d'une base ARDUINO
c) De mettre en œuvre un maître MODBUS sur ce capteur
d) D'écrire un esclave MODBUS en Python permettant de dialoguer avec votre capteur
e) De stocker les données reçues dans une base de données SQLite
f) De présenter les commandes et les données lues sur une IHM en Python (site WEB ou autre)

Ce projet se déroule sur deux séances (14 heures).

En fin des séances, devra être remis par le binôme un rapport de 2 à 8 pages récapitulant :

- le descriptif simple des trames échangées avec le boîtier,
- le principe de fonctionnement de votre esclave MODBUS (programme capteur),
- le principe de fonctionnement de votre maître MODBUS (programme PC),
- le principe de fonctionnement de votre IHM,
- le principe de fonctionnement de votre application globale,
- une conclusion sur les acquis, les difficultés rencontrées, et les perspectives d'amélioration.

2 Appréhender le protocole MODBUS ASCII

2.1 Préambule

2.2 MODBUS : un protocole « Maître-Esclaves »

- Une station est déclarée « Maître » et toutes les autres sont déclarées « Esclaves ».
- Les échanges sont toujours à l'initiative de la station dite « **Maître** ».
- La structure est toujours : « **Question du Maître** » puis « **Réponse de l'esclave interrogé** » :

2.3 MODBUS en version ASCII

La communication MODBUS **ASCII** s'appuie sur le transfert de **caractères** codés en ASCII sur un port série asynchrone.

2.3.1 Emission d'un octet

Emettre un octet consiste à émettre les **deux caractères** représentant sa valeur hexadécimale.

Exemple : émettre l'octet 0111111 (0x7F en hexadécimal et 127en décimal) revient à émettre successivement :
- le caractère 7 (code ASCII = 0x37),
- le caractère F (code ASCII = 0x46).

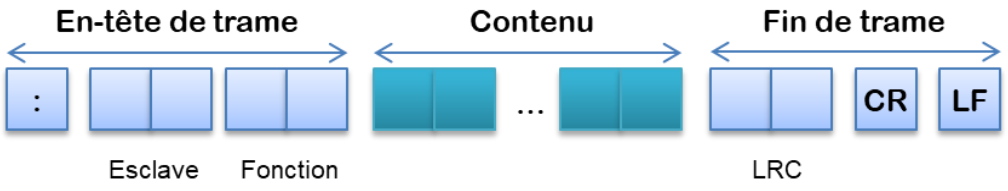
2.3.2 Emission d'un mot (mot de 16 bits)

Emettre un mot consiste à émettre les **quatre caractères** représentant sa valeur hexadécimale.

Exemple : émettre le mot de valeur hexadécimale 0x3A5C revient à émettre successivement :
- le caractère 3 (code ASCII = 0x33),
- le caractère A (code ASCII = 0x41),
- le caractère 5 (code ASCII = 0x35),
- le caractère C (code ASCII = 0x43).

2.4 Structure des trames MODBUS-ASCII

Une trame MODBUS-ASCII a la structure suivante :



- L'**entête** comporte :
  - le caractère « **:** »
  - le **numéro de l'esclave** (sur 1 octet donc 2 caractères) : de 0x01 à 0xFF
  - le **numéro de la fonction** (sur 1 octet donc 2 caractères) (0x 03 ou 0x 04 = lecture ; 0x 06 = écriture)

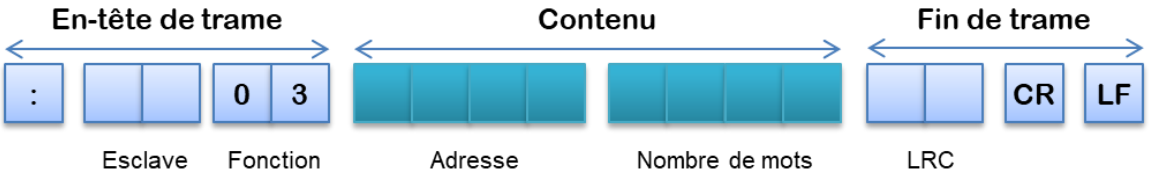
- Le **contenu** est une succession d'octets (donc par paquet de 2 caractères) dont le nombre et la signification sont à interpréter selon la fonction spécifiée dans l'en-tête.

- La **fin de trame** comporte
  - le LRC (sur 2 caractères) : complément à 0x100 (=256) du « checksum » sur 8 bits des différents octets utiles de la trame.
  - le caractère « Carriage Return »
  - le caractère « Line Feed »

2.5 Types de trames MODBUS utilisées par le TP

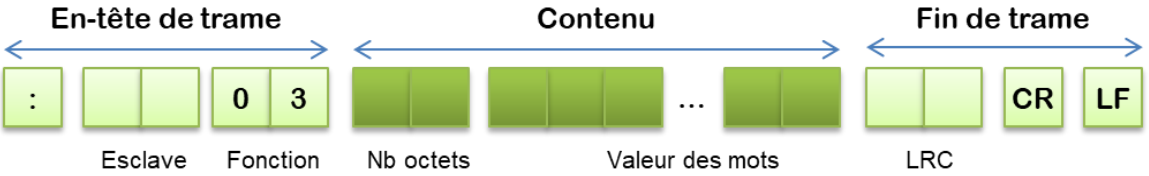
2.5.1 Fonction « 03 » : LECTURE d'un « Holding Register »

QUESTION :



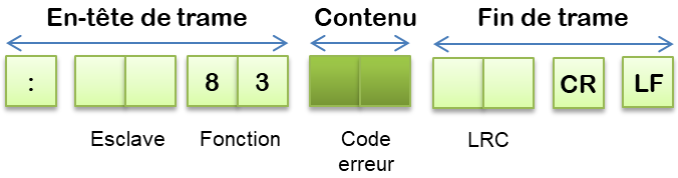
- adresse des données à lire (sur 16 bits donc 2 octets donc 4 caractères)
- nombre de mots de 16 bits (sur 16 bits donc 2 octets donc 4 caractères)

REPONSE OK :



- nombre d'octets lus (double du nombre de mots demandés) (sur un octet donc 2 caractères)
- octets correspondant aux valeurs des mots lus (sur 2 caractères chacun)

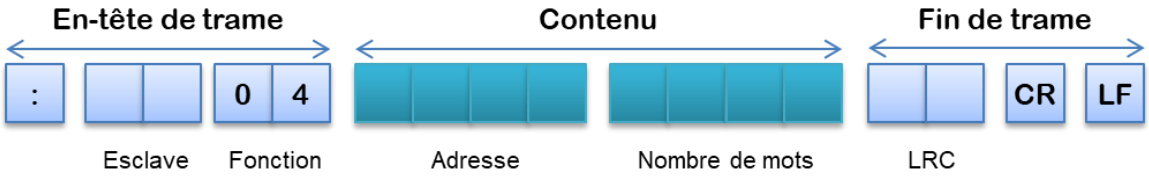
REPONSE ERREUR :



- Code d'erreur ou d'exception (sur un octet donc 2 caractères)

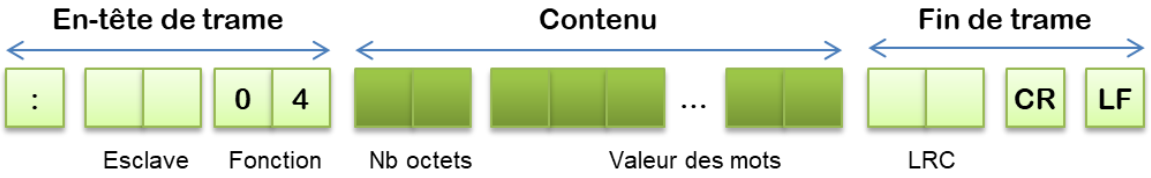
2.5.2 Fonction « 04 » : LECTURE d'un « Input Register »

QUESTION :



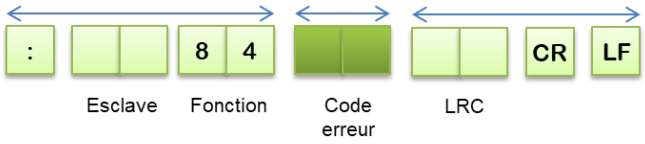
- adresse des données à lire (sur 16 bits donc 2 octets donc 4 caractères)
- nombre de mots de 16 bits (sur 16 bits donc 2 octets donc 4 caractères)

REPONSE OK :



- nombre d'octets lus (double du nombre de mots demandés) (sur un octet donc 2 caractères)
- octets correspondant aux valeurs des mots lus (sur 2 caractères chacun)

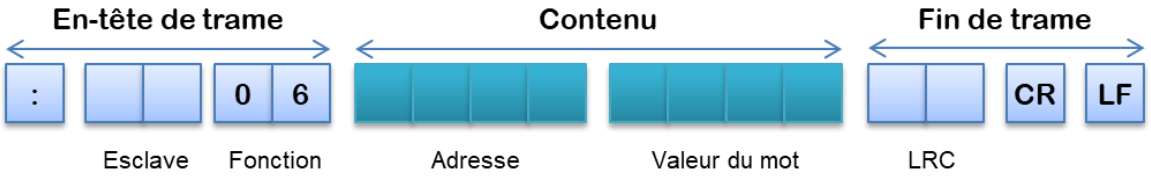
REPONSE ERREUR :



- Code d'erreur ou d'exception (sur 1 octet)

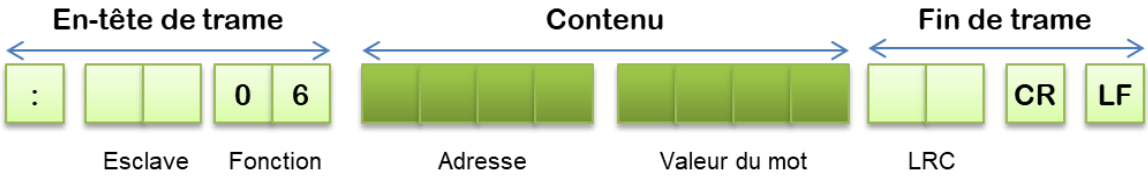
2.5.3 Fonction « 06 » : ECRITURE dans un « Holding Register »

QUESTION :



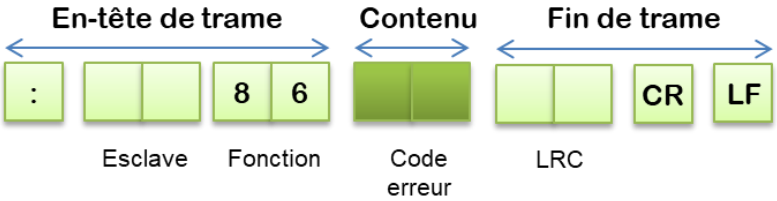
- adresse du registre à modifier (sur 16 bits donc 2 octets donc 4 caractères)
- valeur à écrire dans le registre (sur 16 bits donc 2 octets donc 4 caractères)

REPONSE OK :



- adresse du registre concerné (sur 16 bits donc 2 octets donc 4 caractères)
- valeur écrite dans le registre (sur 16 bits donc 2 octets donc 4 caractères)

REPONSE ERREUR :



- Code d'erreur ou d'exception (sur 1 octet)

3 Matériel : le boîtier « capteur intelligent »

3.1 Description

Il s'agit d'un boîtier comportant une électronique architecturée autour d'un microcontrôleur et d'un capteur de température MCP9800A.

La plage de température de ce capteur est de -55 °C à 125 °C (schéma du capteur en annexe).

3.2 Adressage

Le numéro d'esclave MODBUS de ce boîtier est 1.

Les adresses MODBUS des registres internes au boîtier sont :

	INPUT REGISTERS ***- Lecture = fonction 04***		***HOLDING REGISTERS*** ***- Lecture = fonction 03*** ***- Ecriture = fonction 06***	
Adresse	***Désignation***	***Commentaires***	***Désignation***	***Commentaires***
0	Version boîtier	Format MSB.LSB	Index et Commande	MSB = Index LSB = Commande
1	Mesure de température	voir en annexe l'extrait de documentation du capteur	Bargraphe	0 à 1023
2	Année courante		Année	
3	Jour et mois courants	MSB = Jour LSB = Mois	Jour et mois	MSB = Jour LSB = Mois
4	Heure et minute courantes	MSB = Heure LSB = Minute	Heure et minute	MSB = Heure LSB = Minute
5	Seconde courante	LSB = Seconde	Seconde	LSB = Seconde
6			Période de clignotement	en ms

Remarques :

a) ***MSB*** *= « Most Significant Byte » = octet de poids forts*
LSB *= « Last Significant Byte » = octet de poids faible.*

b) Les ***codes de commande*** *à écrire dans le LSB du Holding Register n°0 sont :*

0x10 = Mise à la date et heure
0x20 = Démarrage mesure
0x30 = Arrêt mesure
0x40 = clignotement lent
0x50 = clignotement rapide
0x60 = clignotement personnalisé

d) pour éviter qu'une même commande soit prise en compte plusieurs fois, on associe à chaque nouvelle commande une nouvelle valeur d'index en l'incrémentant de 1 (MSB du registre Index_Commande).

4 Logiciels

Le TP nécessite des logiciels suivants :

4.1 « Virtual Serial Port Emulator »

Ce logiciel est un programme permettant de créer, tester et déboguer des applications utilisant le port série.

Ce logiciel permet la création d'appareils virtuels pour envoyer et recevoir des données. Contrairement aux ports physiques, les périphériques virtuels peuvent être utilisés plusieurs fois par plusieurs applications. VPSE vous permet de partager les données physiques de plusieurs applications, d'exposer des ports au réseau local, de créer des ports virtuels, etc.

4.2 « Serial Port Monitor »

Ce logiciel sert à observer l'activité d'un port série et dans notre cas, celui sur lequel le PC et le capteur s'échangeront des trames selon le protocole MODBUS ASCII.

Ce logiciel permet en particulier :

- de visualiser les trames émises et reçues par le PC sur son port série
- d'émettre des trames construites manuellement.

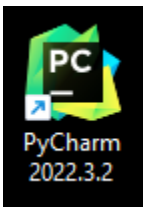
4.3 « Simulide»

Ce logiciel est un simple simulateur de circuit électronique en temps réel pour apprendre et expérimenter des circuits électroniques et des microcontrôleurs simples, prenant en charge PIC, AVR et Arduino.

Ce logiciel permet en particulier

- de créer un montage électronique à base ARDUINO en le programmant
- de tester votre montage
- d'échanger des données via une liaison série avec un autre composant.

4.4 « PyCharm »



PyCharm est un environnement de développement intégré utilisé pour programmer en Python.

Ce logiciel permet l'analyse de code et contient un débogueur graphique. Il permet également la gestion des tests unitaires, l'intégration de logiciel de gestion de versions, et supporte le développement web avec Django.

4.5 « Arduino IDE »

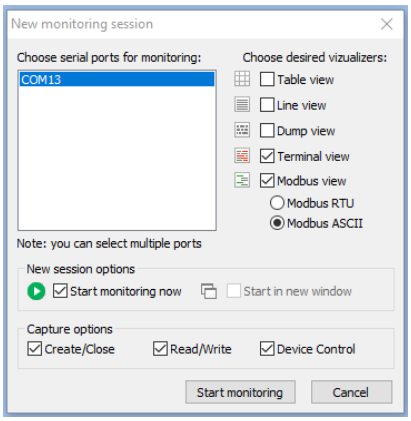
Le logiciel Arduino IDE est un Environnement de Développement Intégré (IDE).

L'Arduino IDE permet :

- d'éditer un programme : des croquis (sketch en Anglais), les programmes sont écrits en langage C
- de compiler ce programme dans le langage « machine » de l'Arduino,
- de téléverser le programme dans la mémoire de l'Arduino,

5 Travail à effectuer

5.1 Mise en service du « Serial Port Monitor »



☞ *Lancer ce logiciel à partir de son icône sur le bureau.*
☞ *Lancer le menu « Session => **New session** ».*
☞ *Vérifier que les cases sont cochées exactement comme dans la fenêtre ci-contre, puis valider ces choix en cliquant sur « Start monitoring ».*

Pour établir la connexion entre ce logiciel et le port série à espionner, il faut « ouvrir » ce port série :

☞ Lancer le menu « View => **Send dialog** ».
☞ Expliquer le rôle des listes déroulantes « Baudrate » et « Parity ».
☞ Vérifier les valeurs suivantes :

Port : celui associé au capteur,	Baudrate : 9600,	Databits : 8
Parity : « No parity »,	Flow control : « None »,	Stopbits : « 1 stop bit »

☞ Cliquer ensuite sur « Open » pour établir la connexion.

***Remarque :** si le bouton Open est grisé, lancer le menu « Help=>Enter registration Code », saisir le nom et le code puis fermer et relancer le logiciel*

Une fois la connexion établie, la grande ligne blanche au centre de cette même fenêtre permettra de saisir la trame que l'on souhaite faire envoyer par le PC vers le boîtier.

5.2 Génération manuelle de trames

On rappelle que dans une trame MODBUS ASCII, le LRC s'obtient par le complément à 0x0100 de l'octet de poids faible de la somme des octets utiles.

Par ailleurs, le tableau ci-dessous indique le code ASCII des caractères utiles ici.

caractère	*code ASCII (Hexa)*
:	*0x3A*
CR	*0x0D*
LF	*0x0A*
0 à 9	*0x30 à 0x39*
A à F	*0x41 à 0x46*

5.2.1 Lecture d'Input Registers

☞ Définir la suite de caractères constituant la trame chargée de lire les 2 « Input Registers » d'adresses 4 et 5.
☞ Convertir cette suite de caractères en hexadécimal (voir tableau ci-dessus).
☞ Inscrire cette trame et l'envoyer (« Send ») vers le boîtier en espérant que celui-ci réponde !

Remarque : on peut renvoyer une trame déjà émise en accédant à la liste déroulante : 

Si aucune réponse du boîtier n'apparaît dans les fenêtres « Modbus View » et « Terminal View », vérifier à chaque tentative d'envoi :

- que le boîtier reçoit des caractères (voyant jaune près du coin)
- que le Modbus View indique un checksum OK et non BAD
- que le boîtier réponde (autre voyant jaune).

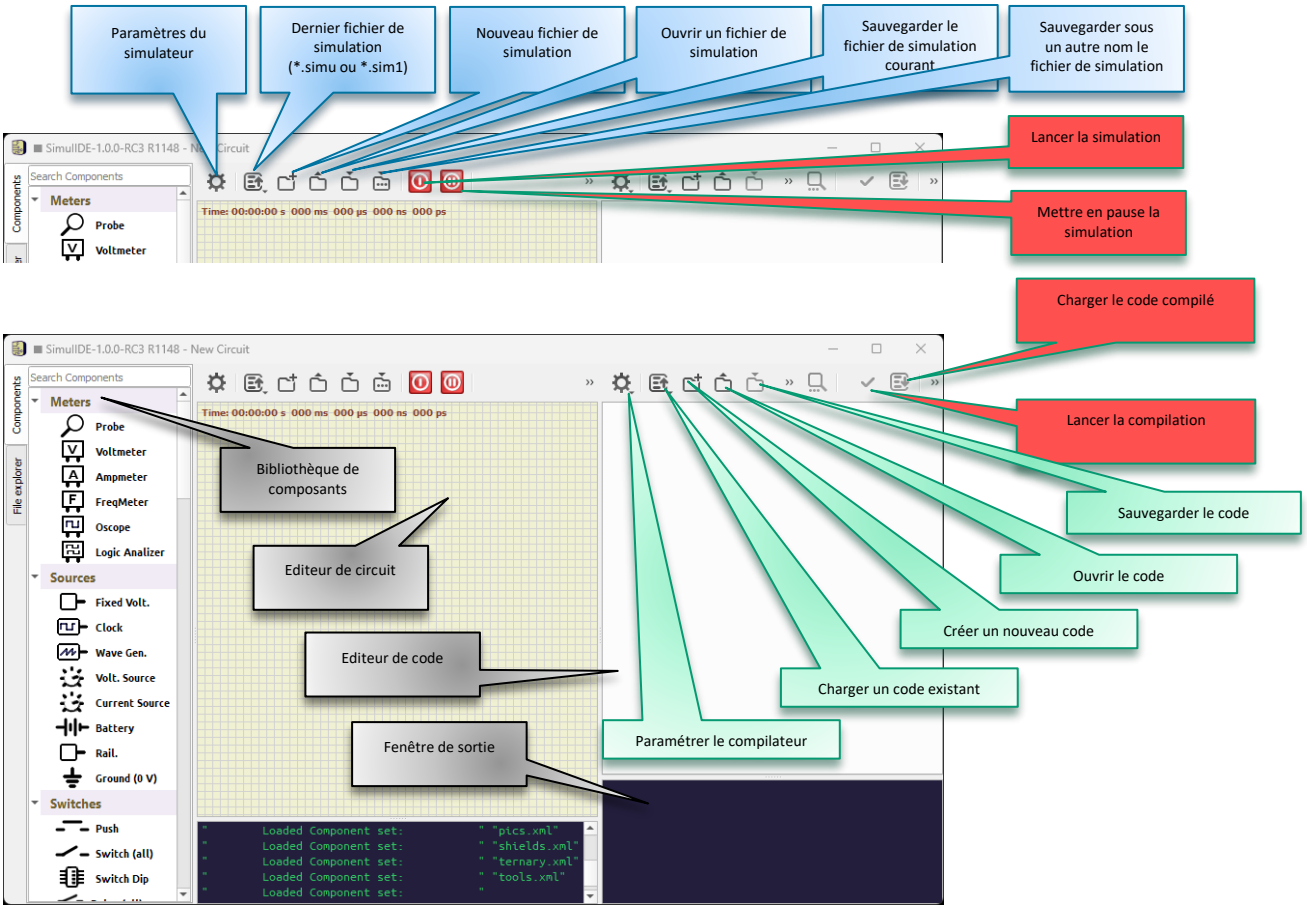
☞ Une fois que tout est OK, relever les trames question et réponse.
☞ A quoi correspond le contenu de la trame réponse et quelles en sont les valeurs ?

5.2.2 Ecriture dans un « Holding Register »

☞ Construire la trame qui permet d'écrire la valeur décimale 255 dans le « Holding Register » d'adresse 1. Interpréter les trames échangées et comparer avec les effets sur le boîtier.
☞ Ecrire la trame qui allume tous les voyants du bargraphe.

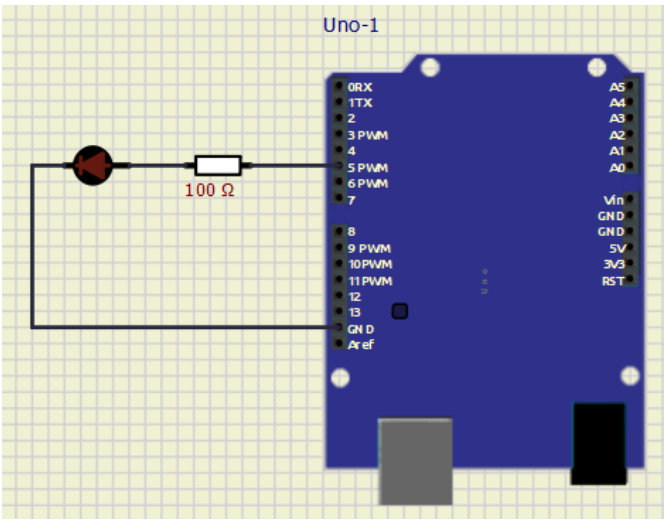
5.3 Prise en main de SimulIDE

☞ Lancer ce logiciel « simulide » à partir de son icône sur le bureau.



5.3.1 Mon premier circuit

☞ Dans la « bibliothèque de composant » (arborescence.de gauche), rechercher « Micro =>Arduino=> Uno » puis glisser le dans l'éditeur de circuit
☞ Ajouter, de la même manière, une Led (Outputs => Leds => Led)
☞ Ajouter, de la même manière, une résistance de 100 Ohms (Passive => Resistors => Resistor)
☞ Relier les différents composants comme indiqué ci-dessous



☞ Sauver votre circuit dans un nouveau sous-répertoire « clignotement_led » de votre répertoire projet sous le nom « clignotement_led.sim1 »
☞ Créer un nouveau code source et sauvegarder le fichier dans le sous-répertoire « clignotement_led » de votre répertoire projet sous le nom « clignotement_led.ino »
☞ Saisir le code ci-dessous

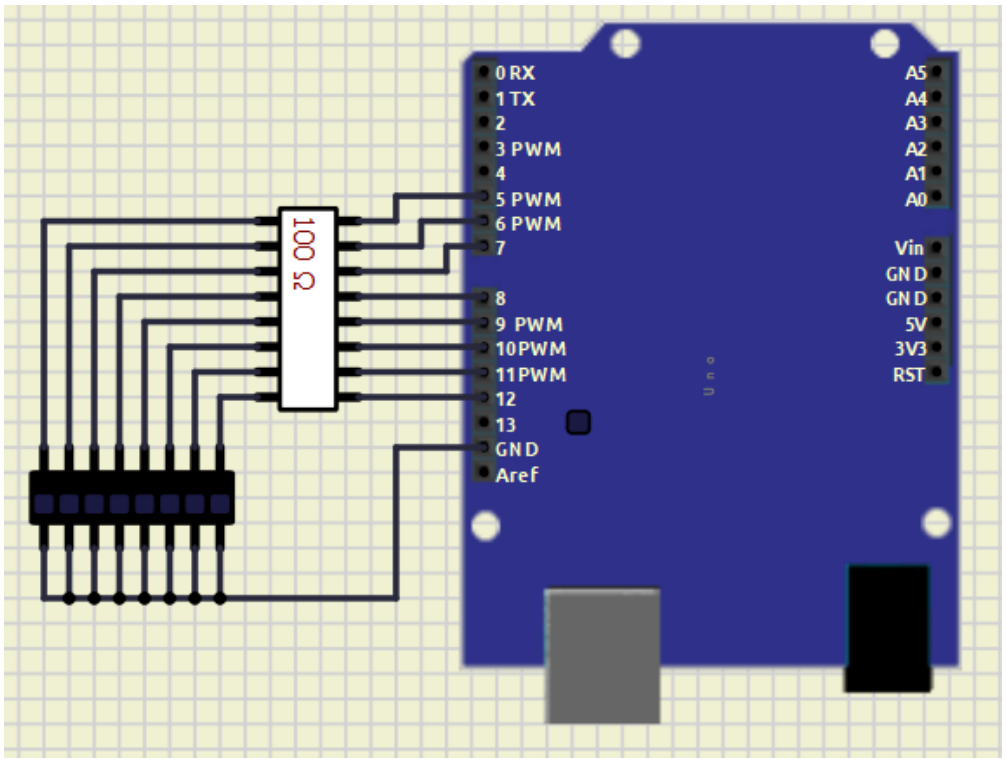
clignotement_led.ino*

```
1  #define PIN_LED 5
2
3  void setup()
4  {
5       pinMode(PIN_LED,OUTPUT);
6  }
7
8  void loop()
9  {
10      digitalWrite(PIN_LED,HIGH);
11      delay(500);
12      digitalWrite(PIN_LED, LOW);
13      delay(500);
14 }
```

☞ Compiler le code le code ci-dessous
☞ Charger le code compile dans la cible
☞ lancer la simulation ➔ La led devrait clignoter ☺

5.3.2 Mise en œuvre d'un bargraphe

☞ Créer un nouveau circuit
☞ Sauver ce dernier dans un nouveau sous-répertoire « Bargraphe » dans votre répertoire de projet
☞ Dans la « bibliothèque de composant » (arborescence.de gauche), rechercher « Micro =>Arduino=> Uno » puis glisser le dans l'éditeur de circuit
☞ Ajouter, de la même manière, un afficheur (Outputs => Leds => LedBar)
☞ Réaliser le câblage et le programme permettant d'afficher un octet sur le bargraphe en vous inspirant de la vue ci-dessous



5.3.3 Mise en œuvre du capteur de température DHT22

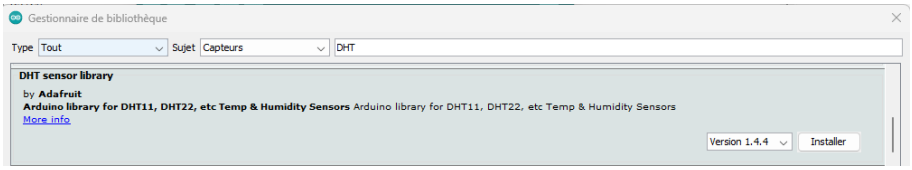
☞ Créer un nouveau circuit
☞ Sauver ce dernier dans un nouveau sous-répertoire « capteur_DHT22 » dans votre répertoire de projet
☞ Dans la « bibliothèque de composant » (arborescence.de gauche), rechercher « Micro =>Arduino=> Uno » puis glisser le dans l'éditeur de circuit
☞ Ajouter, de la même manière, une Led (Outputs => Leds => Led)
☞ Ajouter, de la même manière, une résistance de 100 Ohms (Passive => Resistors => Resistor)
☞ Ajouter, de la même manière, un capteur de température DHT22 (Micro => Sensors => DHT22)

Lire la documentation du capteur sur le site « arduino France : https://arduino-france.site/dht22-arduino/ » pour comment connecter et programmer le DHT22.

☞ Réaliser le câblage et le programme

Remarques :

a) La documentation du capteur DHT22 est disponible sur internet.
b) Inclure la bibliothèque DHT de chez ADAFRUIT vous devez :
   a. Exécuter le logiciel **Arduino IDE**
   b. Dans le menu « Croquis => Inclure une bibliothèque => Gérer les bibliothèques » (*Sketch→Include Library→Manage Libraries…*), rechercher dans « Sujet » **Capteurs** puis rechercher **DHT**



   c. Sélectionner « ***DHT Sensor librairy*** » de chez Adafruit, puis clique sur « *Installer* ». Il vous demandera d'installer aussi « *Adafruit Unified Sensor* » dans ce cas cliquer sur « *Install all* ».

☞ Revenir dans Simulide, puis compiler votre programme, charger et tester.

5.3.4 Mise en oeuvre d'un LCD 16 x 2 non I2C

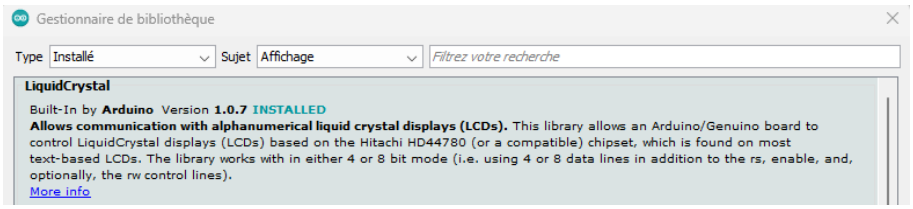
☞ Créer un nouveau circuit
☞ Sauver ce dernier dans un nouveau sous-répertoire « LCD_1602 » dans votre répertoire de projet
☞ Dans la « bibliothèque de composant » (arborescence.de gauche), rechercher « Micro =>Arduino=> Uno » puis glisser le dans l'éditeur de circuit
☞ Ajouter, de la même manière, un afficheur (Outputs => Displays => HD44780)

Remarque : Lire la documentation du capteur sur le site « arduino France : https://arduino-france.site/lcd-1602/ » pour comment connecter et programmer un afficheur LCD sans I2C.

☞ Réaliser le câblage et le programme

Remarque : Inclure la bibliothèque LiquidCrystal de chez Arduino vous devez :

a) Exécuter le logiciel **Arduino IDE**
b) Dans le menu « Croquis => Inclure une bibliothèque => Gérer les bibliothèques » (*Sketch→Include Library→Manage Librairies…*), rechercher dans « Sujet » **Affichage** puis rechercher **LiquidCrystal**



c) Sélectionner « **LiquidCrystal** » de chez Arduino, puis cliquer sur « *Installer* ».

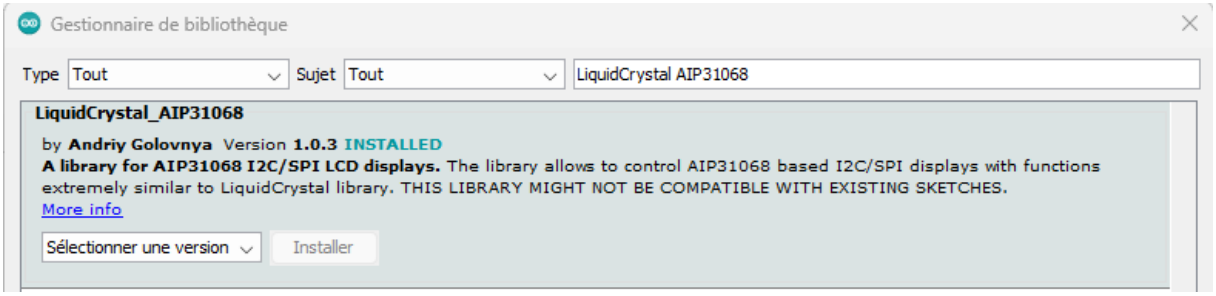
☞ Revenir dans Simulide, puis compiler votre programme, charger et tester.

5.3.5 Mise en oeuvre d'un LCD 16 x 2 I2C

☞ Créer un nouveau circuit
☞ Sauver ce dernier dans un nouveau sous-répertoire « LCD_1602_I2C » dans votre répertoire de projet
☞ Dans la « bibliothèque de composant » (arborescence.de gauche), rechercher « Micro =>Arduino=> Uno » puis glisser le dans l'éditeur de circuit
☞ Ajouter, de la même manière, un afficheur (Outputs => Displays => Aip31068_i2c)

Remarque : Inclure la bibliothèque LiquidCrystal AIP31068 de chez Arduino vous devez :

a) Exécuter le logiciel **Arduino IDE**
b) Dans le menu « Croquis => Inclure une bibliothèque => Gérer les bibliothèques » (*Sketch→Include Library→Manage Libraries···*), rechercher dans « Sujet » **Affichage** puis rechercher **LiquidCrystal AIP31068**



Lire la documentation du capteur sur le site « arduino France : https://arduino-france.site/lcd-1602/#4 » pour comment connecter et programmer un afficheur LCD avec I2C.

☞ Réaliser le câblage et le programme

5.4 Réalisation de votre capteur IdO de mesure de la température

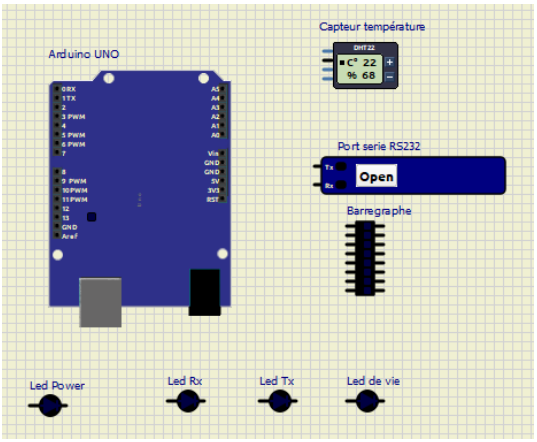
On rappelle que ce capteur est un capteur de température dialoguant sur un port série via le protocole MODBUS.

5.4.1 Réalisation du montage électronique

A l'aide des éléments suivants (repris dans l'image de droite) :

- Arduino UNO
- Capteur de température
- Leds de vie, Rx, Tx, Power
- Barregraphe
- Port série RS232
- Réseaux de résistances 100 Ω (non représentés)
- Résistances 50 Ω (non représentées)

vous devez réaliser le montage de votre capteur IdO.



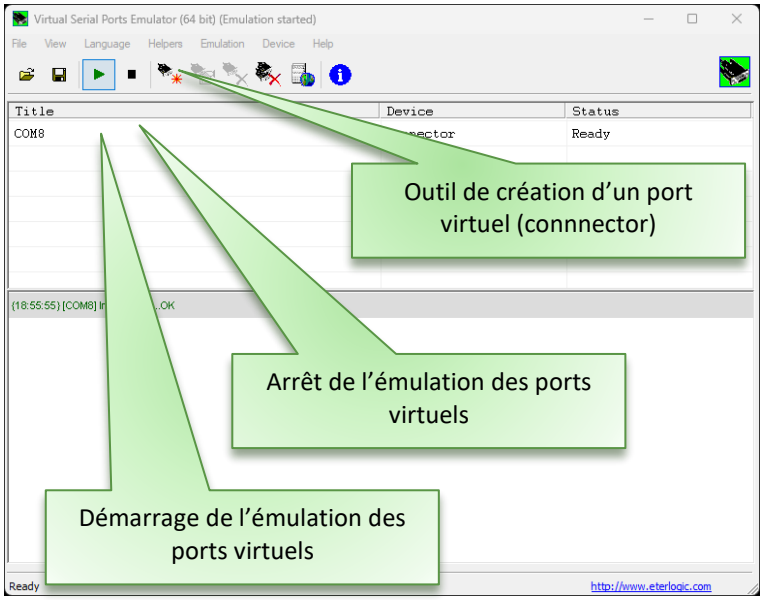
5.4.2 Ecrire le programme d'acquisition de la température

Le but de cette étape est d'écrire un programme ARDUINO UNO qui va réaliser les fonctions suivantes :

a) Faire clignoter la led de vie à la période de 500 ms
b) Lire la température sur le capteur DHT22 à la période de 2000 ms
c) Envoyer la valeur de la température sur le port série COM10.

Pour tester votre serveur, vous devez :

- Créer un port de communication série virtuel à l'aide de l'outil VSPE
- Utiliser ***Serial Port Monitor*** pour échanger des trames MODBUS avec votre capteur.



☞ Ecrire et tester votre programme.

Remarques :

a) *Pour la structure de votre programme, je vous conseille de faire une machine d'état à la période de 100 ms.*

b) *Pour visualiser les chaines envoyées, vous pouvez utiliser « Serial Port Monitor », le moniteur série de Simulide ou le logiciel « Putty » qui est présent sur vos machines.*

5.4.3 Ecrire le programme d'esclave MODBUS

☞ Ecrire le programme Arduino qui permettra de faire de votre IdO un esclave MODBUS.

Pour rappel :

1) le numéro d'esclave MODBUS de votre boîtier est 1.
2) Les adresses MODBUS des registres internes au boîtier sont :

	INPUT REGISTERS - *Lecture = fonction 04*		***HOLDING REGISTERS*** - *Lecture = fonction 03* - *Ecriture = fonction 06*	
Adresse	***Désignation***	***Commentaires***	***Désignation***	***Commentaires***
0	Version boîtier	Format MSB.LSB	Index et Commande	MSB = Index LSB = Commande
1	Mesure de température	voir en annexe l'extrait de documentation du capteur	Bargraphe	0 à 1023
2	Année courante		Année	
3	Jour et mois courants	MSB = Jour LSB = Mois	Jour et mois	MSB = Jour LSB = Mois
4	Heure et minute courantes	MSB = Heure LSB = Minute	Heure et minute	MSB = Heure LSB = Minute
5	Seconde courante	LSB = Seconde	Seconde	LSB = Seconde
6			Période de clignotement	en ms

6 Création de l'IHM

6.1 Utilisation des procédures de librairie serial

Les développements en Python se feront au moyen du logiciel « PyCharm ».

Vous devrez écrire les fonctionnalités suivantes dans votre projet :

- **`connect`** : établir la connexion avec le port série indiqué
- **`disconnect`** : couper la connexion
- **`emettreChaine`** qui émet sur le port série configuré la chaîne de caractères de son choix,
- **`afficherTrameSiRecue`** qui place les caractères reçus par le PC dans un tableau puis les affiche.

Pour cela importer la librairie relative au port série avec le code :

```
import serial
```

Si la librairie n'est pas présente dans votre projet vous pouvez l'ajouter avec la commande suivante dans le terminal de votre projet :

```
pip install pyserial
```

☞ A l'aide de ces procédures, on souhaite à chaque appui de touche, lire dans le boîtier les secondes courantes.

☞ Définir la liste des caractères de la trame qui lit les secondes courantes dans le boîtier.

Remarque : Aide port série :

```
import serial
if __name__ == '__main__':
    print("Demarrage de l'application")
    ser = 0
    try:
        # Connexion au port série
        ser = serial.Serial(
            port='COM1',
            baudrate=9600,
            parity=serial.PARITY_NONE,
            stopbits=serial.STOPBITS_ONE,
            bytesize=serial.EIGHTBITS,
            timeout=3
        )
    except:
        print("Erreur lors de l'ouverture du port série")
        exit(0)
    cmd = ":1103006B00037E\r\n"
    print(cmd)
    cmd = [ord(c) for c in cmd]
    print(cmd)
    print([hex(x) for x in cmd])
    ser.write(cmd)
    ser.readline()
    # ser.read_until(size=??)
    # ser.read_until(expected=??)
    ser.close()
```

6.2 Mise au point de la procédure « EmettreQuestion »

On souhaite disposer d'une procédure universelle « **`emettreQuestion`** » permettant d'émettre n'importe quelle trame question en ne lui fournissant que les données utiles.

☞ Quelles données cette procédure devra-t-elle calculer ?
☞ Déterminer les paramètres à lui passer et leur type.
☞ Ecrire la procédure « emettreQuestion » votre code python.

6.3 Lecture/écriture de registres

6.3.1 Lecture de la valeur de la température

La procédure « **`afficherTrameSiRecue`** » place les caractères reçus dans un tableau « **`trameRecue[0...50]`** » et les affiche.
La variable « **`flagTrameSiRecue`** » indique qu'une trame est arrivée.
La variable « **`nbCarRecu`** » indique le nombre de caractères reçus (CR et LF inclus).

☞ Dans votre code python écrire et tester la procédure « lireTemperature » chargée de lire la température et sa date d'acquisition puis de les afficher sur l'écran du PC.

Remarque : la procédure « **`afficherTrameSiRecue`** » dispose d'un paramètre booléen définissant si la trame brute doit être affichée ou non (true : affichage de la trame brute, false : pas d'affichage de celle-ci).

6.3.2 Ecriture de la valeur des secondes sur le Bargraphe

☞ Ecrire et tester une procédure qui visualise les secondes écoulées sur le bargraphe.

Remarque : Entre deux échanges, laisser un délai d'une seconde à l'aide de la fonction « sleep(1) » à l'aide de la librairie « time ».

6.3.3 Ecriture de la température sur le Bargraphe

☞ Ecrire et tester une procédure qui visualise la Température sur le bargraphe.

6.4 Ecriture de commandes

☞ Ecrire une commande consiste à écrire deux octets dans le « Holding Register » n°0 :

- le MSB contient l'index qui doit être incrémenté à chaque nouvelle commande envoyée
- le LSB contient le code de la commande à effectuer :
  - 0x10 = Mise à la date et heure écrites dans les « Holding Registers » n°2 à 5
  - 0x20 = Démarrage mesure
  - 0x30 = Arrêt mesure
  - 0x40 = clignotement lent
  - 0x50 = clignotement rapide
  - 0x60 = clignotement personnalisé à la période écrite dans le « Holding Register » n°6

6.4.1 Procédures « clignoter_lent » et « clignoter_rapide »

☞ Faire en sorte que l'appui sur la touche « L » provoque un clignotement lent et que la touche « R » provoque un clignotement rapide.

6.4.2 Procédure « mise à jour de la date et heure »

☞Faire en sorte que l'appui sur la touche « M » provoque la mise à jour de la date et de l'heure dans le boîtier.
Indications :

- Il faut d'abord recopier la date et heure du PC dans les « Holding Registers » correspondants avant de lancer une commande de mise à la date et à l'heure.
- L'appel de « `datetime.now` » permet de récupérer une partie de la date ou de l'heure courantes du PC via la librairie « `datetime` ».
- Respecter un délai entre deux échanges consécutifs

6.4.3 Visualisation des défauts

Nous voulons traduire un défaut ou un arrêt de la mesure sur le capteur par un changement de la vitesse de clignotement (lent, rapide ou personnalisé).

☞ Ecrire la ou les procédures permettant d'arrêter ou de démarrer la mesure dans le capteur.
☞ Ecrire la procédure permettant de détecter un défaut dans le capteur (écart trop important entre deux mesures).
☞ Ecrire la procédure permettant de faire varier la période du clignotant.

6.5 Mise en place d'un service WEB

Suite à la mise en place de la communication ModBus avec votre système IdO, comme vu dans le cours, deux besoins principaux restent à mettre en place :

☞ Stockage des données reçues : Stocker les données reçues dans une base de données SQLite
☞ IHM : Présenter les commandes et les données lues sur une IHM en Python (site WEB ou autre)

6.5.1 Mise en service d'un base SQLite

Vous utiliserez une base de données de type SQLite pour cet exercice.

La librairie « sqlite3 » devrait déjà être présente sur votre installation, ce package est maintenant associé nativement aux versions de python supérieur à python 2.5.

Importez la librairie dans votre projet avec la ligne suivante :

```
import sqlite3
```

Votre base de données SQLite existera sous un format de fichier « .db ». Vous pourrez éditer cette base directement avec du code python ou si vous le souhaitez, vous pouvez utiliser l'utilitaire « *DB Browser for SQLite* ».

En python, si vous initiez une connexion à une base de données (fichier .db) qui n'existe pas celle-ci sera créée. Par exemple nous pouvons créer la base avec une nouvelle connexion telle que :

```
conn = sqlite3.connect('IOT.db')
```

Suite à la création de notre base de données, il est nécessaire de créer une table qui servira au stockage d'information de notre système, à sa configuration, à des données de connexion, des chemins de fichiers… Pour cela, nous devons créer un curseur dans la table et exécuter des requêtes SQL standart tel que :

```
    c = conn.cursor()
    # Creation de la table
    c.execute('''CREATE TABLE Animal (Espece text, Couleur text, Zone_Geographique text,
Recensement real)''')
    # Insertion de données dans la table
    c.execute("INSERT INTO Animal VALUES ('Renard','roux','Europe-Est',621)")
    # Commit (sauvegarde/application des modifications)
    conn.commit()
    # Fermeture de la connexion
    # Attention, toute action sans commit sera perdue
    conn.close()
```

***REMARQUE IMPORTANTE* : Il conviendra de modifier la table et les données pour s'adapter au sujet.**

Si vous souhaitez effectuer une requête dans votre base afin de récupérer des données, une des méthodes est la suivante :

```
    param = ('Renard',)
    c.execute('SELECT * FROM Animal WHERE Espece=?', param)
    print (c.fetchall())
```

La méthode « `fetchall` » récupère toutes les lignes de la table correspondante à la requête. Vous pouvez effectuer une itération sur cette méthode pour traiter les lignes tel que :

```
    resultat = c.fetchall()
    print("NB lignes:  ", len(resultat))
    print("Pour chaque lignes")
    for ligne in resultat:
        print("Espece : ", ligne[0])
        print("Couleur : ", ligne[1])
        print("Zone_Geographique : ", ligne[2])
        print("Recensement : ", ligne[3])
        print("\n")
```

☞créer une base permettant d'historiser les données du capteur (données des « Input Registers »).

Ces données devront obligatoirement être datées avec la meilleure configuration possible (datation par le capteur – *date capteur* - ou datation par le serveur – *date serveur* -). Donc en conséquence vous devez aussi avoir un champ correspondant à la date et l'heure du système choisi.

REMARQUE : Le champ précédent correspondant à la date et l'heure pourra être une clef primaire de votre table, vous êtes libre d'organiser votre ou vos tables de la façon que vous jugerez le plus efficace.

☞Vous devrez mettre en place des fonctions permettant :

- L'insertion de nouvelles données provenant du capteur
- Récupérer tous les X premiers ou derniers points de données
- Récupérer tous les points de données compris entre deux dates
- Récupérer tous les points de données selon un paramètre de température (intervalle ou seuil)
- Récupérer tous les points de données selon une version de capteur.
- Mettre en place en Python d'une méthode de backup de la base de données toutes les heures en Python

6.5.2 Mise en place d'une IHM en python

Afin de pouvoir interagir avec votre système sans commande dans un terminal et pour visualiser les résultats plus simplement, une interface homme machine doit être mise en place.

☞Pour cela deux solutions s'offrent à vous :

- Un WEB serveur et son site WEB, avec le package « *flask* »
- Une interface graphique type formulaire, avec le package « *tkinter* »

Chaque méthode à des avantages et inconvénient qui lui sont propres, dépendant de l'utilisation que vous souhaitez faire dans votre interface et des méthodes d'intégrations de vos fonctions.

Pour chaque méthode, vous devrez mettre en place toutes les fonctions relatives à la base de données, ajouter un affichage sous format de courbe et un export/téléchargement de fichier de mesure.

La solution WEB serveur

Afin de mettre en place notre serveur, nous utiliserons le package « *flask* ». Le package « *Django* » peut aussi répondre à vos besoins si vous avez déjà de l'expérience avec ce package, les principes de fonctionnement étant les mêmes toutefois les exemples de code que vous trouverez dans ce document feront référence à « *flask* ».

Flask nécessite une version de python supérieure à 3.7, les machines de cours que vous utilisez ont au moins la version 3.8.8 d'installée.

Si vous travaillez sur votre machine personnelle la commande d'installation est la suivante :

```
pip install Flask
```

Ajouter l'import suivant pour avoir accès à la majorité des fonctions nécessaires pour cet exercice

```
from flask import Flask, redirect, url_for, request, render_template, session,abort
```

Dans la zone de déclaration de vos constantes et variables globales associez le serveur flask à votre application.

```
app = Flask(__name__)
app.debug = True
if __name__ == '__main__':
    app.run()
```

Vous pouvez démarrer votre serveur avec la commande suivante dans le terminal :

```
flask run -h 10.2.29.xxx -p 5000
```

xxx = représente le dernier octet de l'adresse IP de votre machine

Cette commande démarrera votre serveur sur l'adresse IP indiquée sur le port 5000.

Vous devez obtenir un résultat suivant :

```
Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a
production WSGI server instead.
 * Running on http://192.168.1.15:80
Press CTRL+C to quit
```

Utiliser la commande CTRL+C (sous Windows) pour arrêter votre serveur.

Votre serveur WEB est configuré, vous devez maintenant y ajouter des « routes » qui serviront à l'interaction entre l'utilisateur et le serveur. Nous allons créer une première route paramétrée.

Ajoutez le code suivant à votre application :

```
@app.route('/Bonjour')
def index():
    return 'Bienvenue sur votre serveur WEB !'
```

Démarrez votre serveur et via un navigateur WEB, saisissez l'URL suivante « `10.2.29.xxx/Bonjour` ».

Cette « route », vous retourne une chaîne de caractère sans prendre de paramètre et non formattée pour de l'affichage HTML.

Vous pouvez récupérer des paramètres par la méthode GET avec une route suivante :

```
@app.route('/Addition',methods = ['POST', 'GET'])
def Addition():
    request.args.get('a')
    request.args.get('b')
    # retourne a + b
    return "Le résultat est "
```

☞Modifiez le code suivant pour obtenir une route additionnant deux entiers.

☞Vous pouvez tester cette méthode avec la requête URL telle que : http://10.2.29.xxx/Addition?a=1&b=5

Vous savez maintenant effectuer des requêtes simples sur votre serveur WEB, toutefois celui-ci nécessite toujours l'écriture d'URL qui sont prônes à erreur, une interface graphique type HTML est préférable.

☞*Créez un dossier « templates » (ATTENTION : ne pas changer le nom) dans le répertoire de votre projet python, vous allez placer vos pages HTML dans ce dossier.*

☞Ajoutez une nouvelle route telle que :

```
@app.route('/TPtemplate')
def hello():
    return render_template('index.html')
```

☞Créez le fichier index.html dans votre dossier de *templates* tel que :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>FlaskWebServeur</title>
</head>
<body>
    <h1>Bienvenue!</h1>
    <h2>Votre premier site WEB fonctionne !</h2>
</body>
</html>
```

Démarrer votre serveur et accédez à l'URL associée à cette route, vous devez obtenir votre page HTML

***Remarque** : Lors de l'édition de page HTML, Pycharm vous propose une preview de la page dans Pycharm ou en utilisant un navigateur installé sur votre machine, n'hésitez pas à utiliser cette fonction !*

Cette page HTML n'affiche que du texte, nous allons lui rajouter un paramètre d'exécution, la date et heure du serveur.

☞Modifiez l'appel à votre page de la façon suivante :

```
return render_template('index.html', date_heure=datetime.datetime.now())
```

☞Modifiez votre page HTML avec un nouveau champs :

```
<h3>{{ date_heure }}</h3>
```

Vous savez maintenant accéder à une page web et utiliser des variables. Si vous souhaitez afficher des logos, bannières, pieds de pages, tableau dynamique…il conviendra d'effectuer un retour de fonction en concaténant plusieurs chaines de charactères qui formeront la page complète.

De façon la plus simple, traitez votre page web comme une succession de bloc de code HTML dont le retour se fait en une fois, par exemple :

```
Return render_template(header.html') + str(codeHTMLdynamique) + render_template(footer.html')
```

La partie « *codeHTMLdynamique* » pourrait être une chaîne étant le résultat d'une fonction effectuant une requête SQL telle que :

```
<table>
  <tr>
    <th>Espece</th>
    <th>Couleur</th>
    <th>Zone_Geographique </th>
    <th>Recensement  </th>
  </tr>
  <tr>
    <td>Renard</td>
    <td>roux</td>
    <td>Europe-Est</td>
      <td>621</td>
  </tr>
  <tr>
```

```
    <td>Renard</td>
    <td>Blanc</td>
    <td>Alaska</td>
      <td>875</td>
  </tr>
</table>
```

Avec les champs remplis en itérant sur la réponse SQL. Le formatage (style, police, couleur…) des pages sera à votre discrétion.

L'ajout d'un bouton sur une page peut se faire de la façon suivante.

```
    <form action="/Addition">
      <label for="a">Valeur de a:</label>
      <input type="text" id="a" name="a"><br><br>
      <label for="b">Valeur de b:</label>
      <input type="text" id="b" name="b"><br><br>
      <input type="submit" value="Submit">
    </form>
```

Cette méthode vous permet de renseigner les deux paramètres pour votre page d'addition et effectuer un « *Submit* » avec un bouton. Les champs « *date time picker* » reprennent des méthodologies similaires.

Pour l'affichage de courbe, le package « *matplotlib* » sera à utiliser. Vous pouvez l'importer de la façon suivante :

```
from matplotlib import pyplot as plt
```

Il est recommandé de générer une image de courbe à partir de *matplotlib* et ensuite d'afficher cette image sur la page web. N'essayez pas de générer des courbes dynamiques.

☞ Placez vos images (coubres, logos ou autres) dans un dossier « *static* » sur la racine de votre projet *(ATTENTION : ne pas changer le nom)*. Vous pouvez afficher une image avec la méthode HTML suivante :

```
<img src="{{url_for('static', filename='courbes.png')}}" title="My plot" />
```

Les données à générer pour le téléchargement devront être au format CSV. Vous pouvez vous inspirer de la méthode suivante pour générer un lien de téléchargement de fichier :

```
<a class="btn btn-primary" role="button" href="{{url_for('static', filename='data.csv')}}"
download="data.csv">Download</a>
```

La solution interface graphique par tkinter

Tkinter est un framework d'interface graphique utilisateur (GUI) pour python, vous l'utiliserez pour mettre en place une interface similaire au serveur WEB. Tkinter est installé de base avec python mais vous devez importer la librairie.

```
Import tkinter
from tkinter import *
```

Remarque : ne pas oublier de commenter l'appel « app.run() » de votre code ou votre service WEB sera démarré.

☞Créez une première fenêtre simple avec le code suivant :

```
fenetre = Tk()
label = Label(fenetre, text="Bonjour - TP IOT")
label.pack()
fenetre.mainloop()
```

***Remarque** : Vous pouvez exécuter votre code avec la commande « python .\VotreFichier.py ».*

☞Exécutez votre programme pour valider son fonctionnement.

Pour ajouter un bouton la méthode est la suivante :

```
Monbouton = tkinter.Button(fenetre, text="MonPremierBouton", command=ButtonCallBack)
Monbouton.pack()
```

La fonction « ButtonCallBack » sera à définir selon vos besoins.

Dans le cas d'une *messageBox* il vous faudra importer :

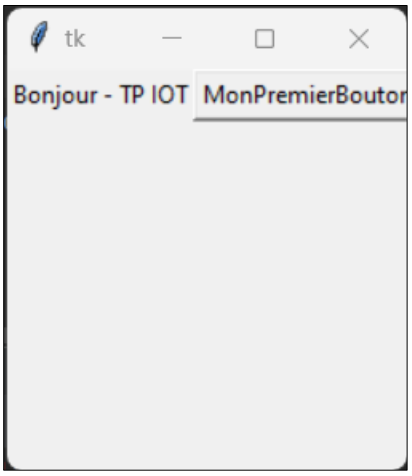
```
from tkinter.messagebox import *
```

Et ajouter la fonction :

```
def ButtonCallBack():
   tkinter.messagebox.showinfo( "Bonjour !", "Texte de message box !")
```

***Attention** : Si nous oubliez de faire appel à la méthode « pack » d'un objet, il faudra préciser son placement*

Tkinter vous permet de gérer le positionnement d'objet soit par une grille ou par leur position « X,Y ». Exemple en « X,Y » :




```
Monbouton.place(x=25, y=100)
# Monbouton.pack()
```

Le bouton ci-dessus sera hors de la fenêtre, vous pouvez modifier la taille de la fenêtre de la façon suivante :

```
fenetre.geometry("200x200")
```

Le placement par grille se fait de la manière suivante :

```
    label.grid(row=1, column=1)
    Monbouton.grid(row=1, column=2)
```

Ne pas oublier de retirer l'appel « pack » si vous utilisez les grilles

L'utilisation d'un placement par grille ou en coordonnées est laissée à votre discrétion.

Si une colonne ou ligne n'est pas utilisée dans le placement par grille alors celle-ci est ignorée et des décalages peuvent avoir lieu, essayez le code précédent avec des numéros ne commencent pas par 1 ou avec des intervalles entre éléments.

Le code suivant vous permettra de mettre en place un simple interface avec *textbox*, bouton et appel de fonction :

```
def testinput():
    INPUT = Entree.get("1.0", "end-1c")
    print(INPUT)
    if(INPUT == "100"):
        Sortie.insert(END, 'Réponse correcte')
    else:
        Sortie.insert(END, "INCORRECT")

if __name__ == '__main__':
    fenetre = Tk()
    fenetre.geometry("300x300")
    fenetre.title(" Multiplication ")
    Question = Label(text="25 * 4 ? ")
    Entree = Text(fenetre, height=10,width=25,bg="RosyBrown1")
    Sortie = Text(fenetre, height=5,width=25,bg="beige")
    Affichage = Button(fenetre, height=2,width=20,text="Show",command=lambda:
testinput())
    Question.pack()
    Entree.pack()
    Sortie.pack()
    Affichage.pack()
    fenetre.mainloop()
```

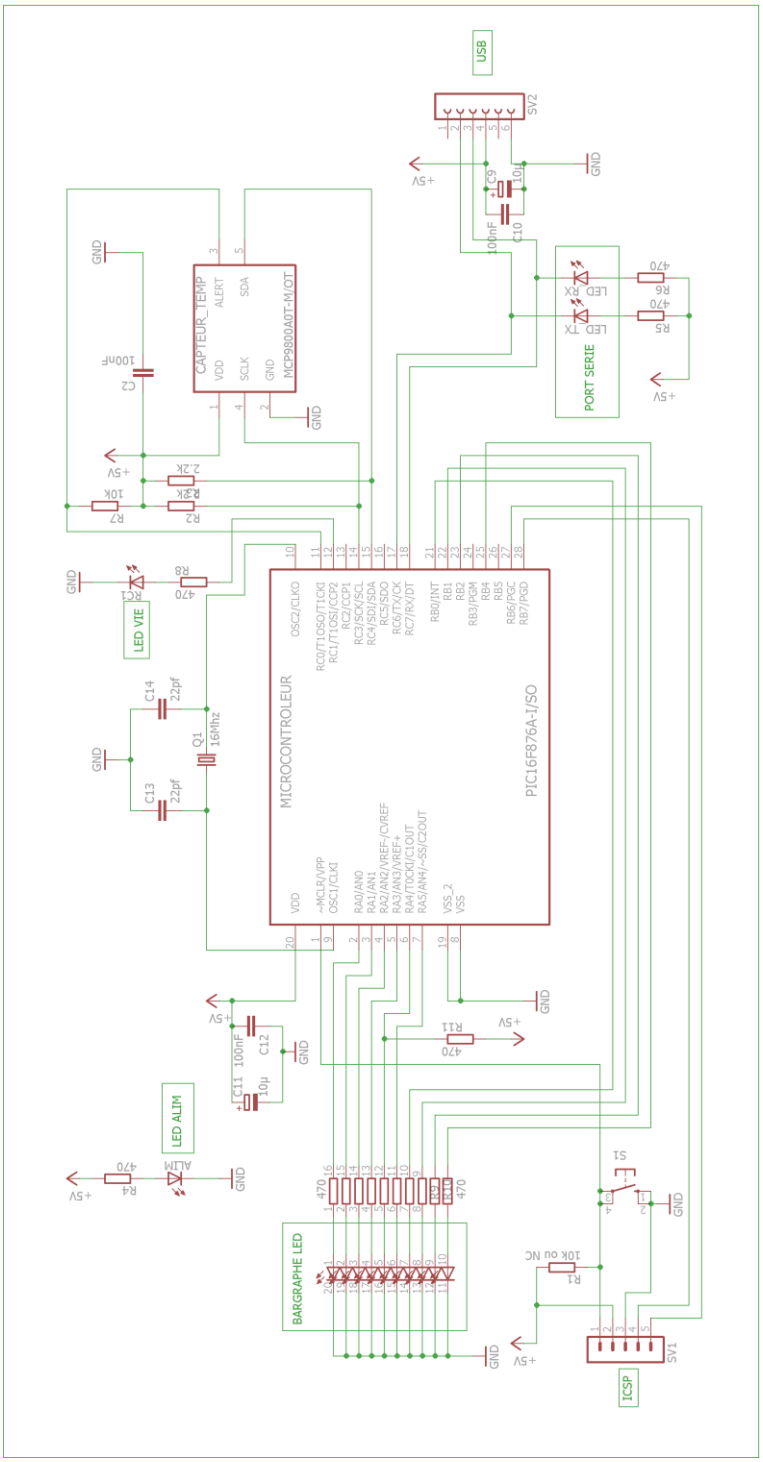
Les deux paramètres de la fonction « *get* » pour le champ texte sont l'indice de démarrage, première ligne, et le charactère à lire en dernier « *end-1c* » correspond au caractère de fin de ligne supprimé. « *end-2c* » supprimerait un caractère en plus.

☞ Mettez en place des fonctionnalités similaire à votre serveur WEB (filtre par mesure, dates, affichage de tableaux…)

Au lieu de télécharger un fichier, vous ouvrirez le répertoire de destination ou ouvrirez le fichier avec un appel « os » sur un éditeur de fichier/tableur.

7 ANNEXES

7.1 Schéma du boîtier



7.2 Extrait de la documentation du capteur MCP9800A

Vous trouverez ci-dessous un extrait de la documentation du capteur qui vous permettra de décoder la valeur de la température ambiante en sachant que le capteur a été paramétré avec la meilleure précision disponible (c'est-à-dire à 2^{-4} °C près).

MCP9800/1/2/3

5.3.1 AMBIENT TEMPERATURE REGISTER (T_A)

The MCP9800/1/2/3 has a 16-bit read-only Ambient Temperature register that contains 9-bit to 12-bit temperature data. (0.5°C to 0.0625°C resolutions, respectively). This data is formatted in two's complement. The bit assignments, as well as the corresponding resolution, is shown in the register assignment below.

The refresh rate of this register depends on the selected ADC resolution. It takes 30 ms (typical) for 9-bit data and 240 ms (typical) for 12-bit data. Since this register is double-buffered, the user can read the register while the MCP9800/1/2/3 performs Analog-to-Digital conversion in the background. The decimal code to ambient temperature conversion is shown in Equation 5-2:

EQUATION 5-2:

$$T_A = \text{Code} \times 2^{-4}$$

Where:

T_A = Ambient Temperature (°C)
Code = MCP9800 output in decimal

REGISTER 5-2: AMBIENT TEMPERATURE REGISTER (T_A) – ADDRESS <0000 0000>b

Upper Half:							
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
Sign	2^6 °C	2^5 °C	2^4 °C	2^3 °C	2^2 °C	2^1 °C	2^0 °C
bit 15							bit 8

Lower Half:							
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
2^{-1} °C/bit	2^{-2} °C	2^{-3} °C	2^{-4} °C	0	0	0	0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note 1: When the 0.5°C, 0.25°C or 0.125°C resolutions are selected, bit 6, bit 7 or bit 8 will remain clear <0>, respectively.