**DOSSIER D’UTILISATION DE LA STATION GEOTHERMIQUE**

**/!\ Mode d’emploi /!\ :**

**Pour comprendre l’utilisation de la station, veuillez-vous référer à la partie interface et gestion de la station page**

**TOUT LE CODE DU PROJET EST ACCESSIBLE SUR LE LIEN GITHUB SUIVANT :**

**https://github.com/1l1ann/Geothermy-Station/tree/master**

***Sommaire :***

Matériel……………………………………………………………………………………………………………page 2

Schéma électrique…………………………………..…………………………………..…………………..page 4

Mesure de température…………………………………..………………………………………………page 5

Gestion de l’horloge…………………………………..…………………………………………………….page 6

Transmission des données sur Google Sheet via Ethernet ……………………………….page 7

Stockage des données sur carte SD…………………………………………………………………..page 11

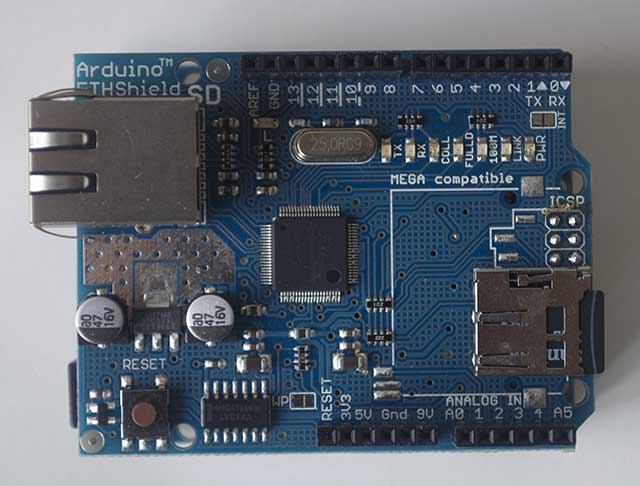
Communication des données par ondes radio FM……………………………………………page 12

Affichage sur écran LCD……………………………………………………………………………………page 13

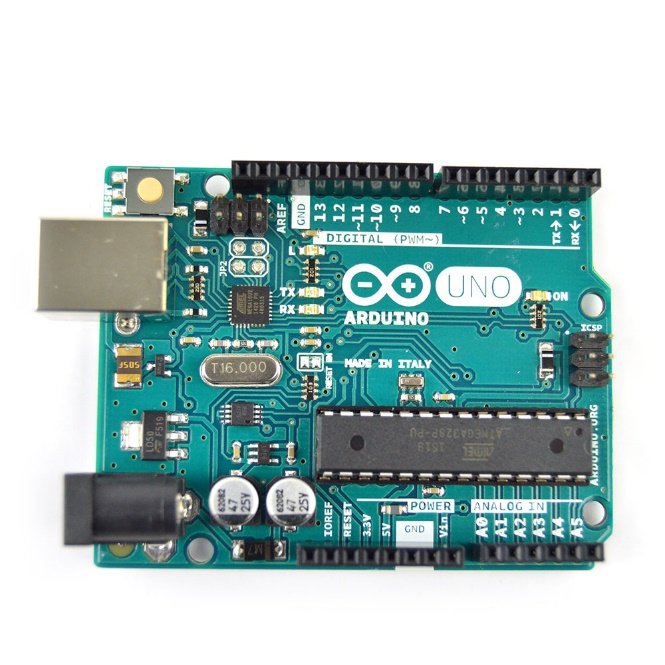
Interface et gestion de la station….…………………………………………………………………page 14

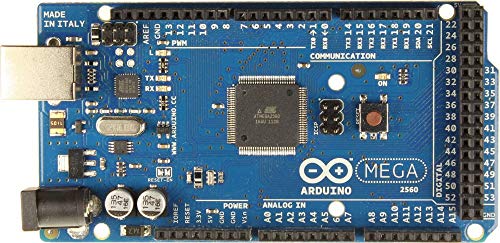
**Matériel**:

* 1 x Carte Arduino MEGA 2560
* 1 x Carte Arduino UNO
* 1 x Shield Ethernet/micro SD
* 2 x Processeur Tiny RTC
* 1 x Ecran LCD
* 1 x Emetteur FM 433MHz
* 1 x Récepteur FM 433MHz
* 1 x carte SD
* 4 x capteur de température protocole I2C (DS18B20)
* 1 x Résistance de 4,7kΩ
* 1 x Résistance de 220Ω
* 1 x Résistance variable
* 1 x Breadboard
* 2 x Antennes de 17cm environ
* Câbles de couleurs
* Fer à souder
* Etain



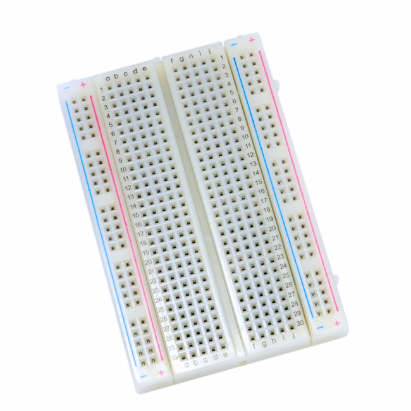
*Shield Ethernet SD*

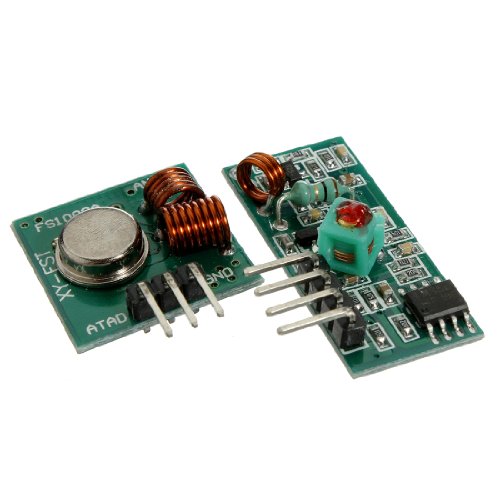




*Carte arduino mega2560*

*Carte arduino uno*

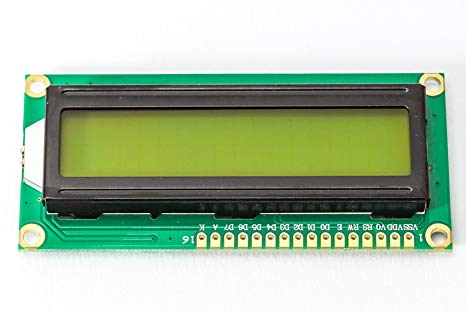




*Breadboard*



*Emetteur/récepteur 433MHz*



*Ecran LCD 16x2*



*Tiny RTC (module calendrier)*



*Petits composants :*

* *Résistances*
* *LED*
* *Câbles M/M, M/F, F/F*
* *Résistances variables*

*Capteur de température I2C*

**Schéma électrique du projet :**

Arduino MEGA

2560

Branchements de la partie de la station « émettrice »

Pin 14

Pin 21

Pin 7

Pin 20

SDA

SDA

SDA

SDA

Sensor

Sensor

Sensor

Sensor

SDA SCL

Tiny RTC

SDA

transmitter

Pin 30

Pin 31

GND +5V

GND +5V

GND +5V

GND +5V

GND +5V

GND +5V

GND

+5V

*4,7kΩ*

Button A

Branchements de la partie de la station « réceptrice »

Button B

Ecran LCD

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Arduino UNO

Pin 12

Pin 11

Pin 2

Pin 3

Potentiomètre

Pin 4

PinA55

Pin 5

Pin A4

SDA

Pin 7

SDA SCL

Tiny RTC

Receiver

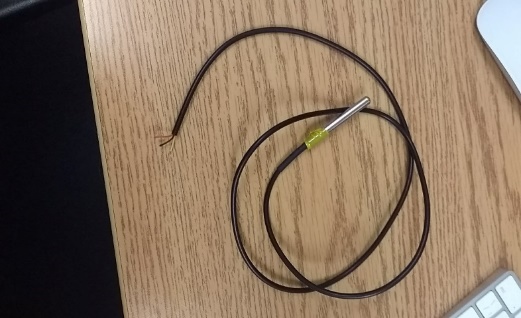
GND +5V

GND +5V

*220Ω*

GND +5V

**Mesure de température :**

Cette partie va permettre l’utilisation des capteurs de température sous le protocole I2C. Les capteurs sont constitués de 3 fils : masse, alim +5V, et data. Le capteur est adressable, on peut donc récupérer une température parmi une infinité de capteur tant qu’on possède l’adresse. C’est là où intervient la librairie Arduino OneWire, qui permet d’effectuer cette action, et ce en ayant à brancher un fil « data » seulement sur une pin digitale.

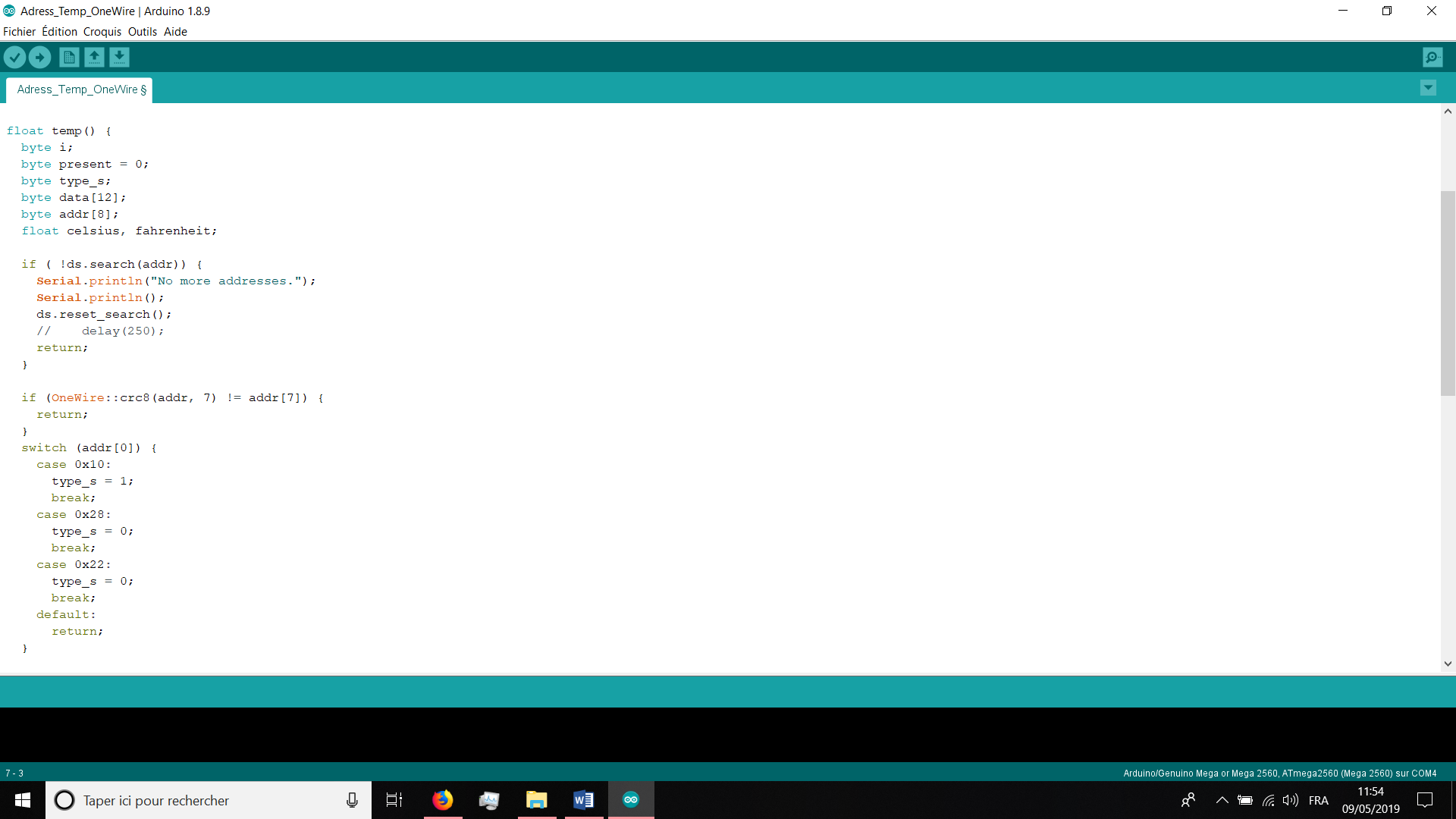
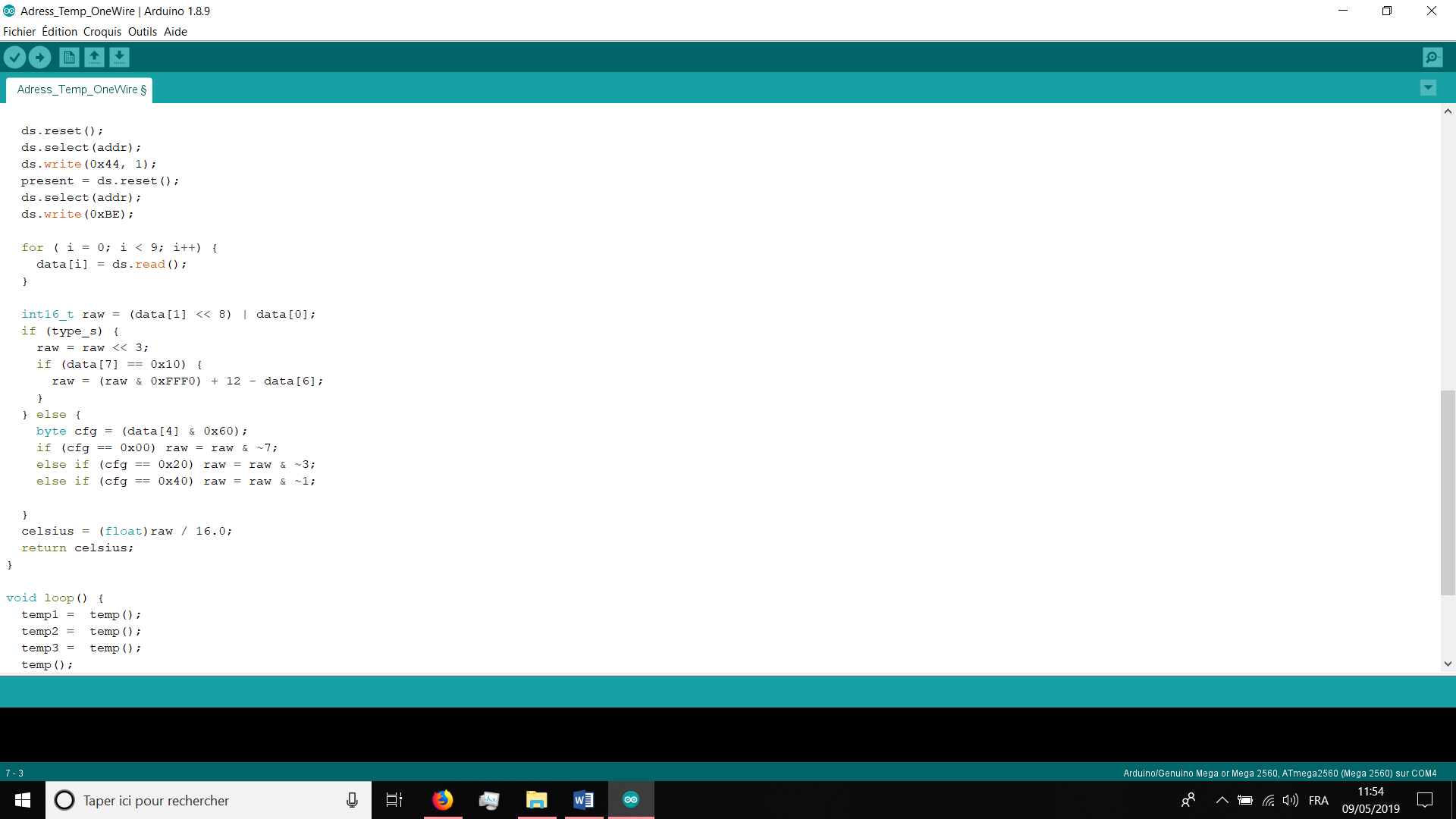
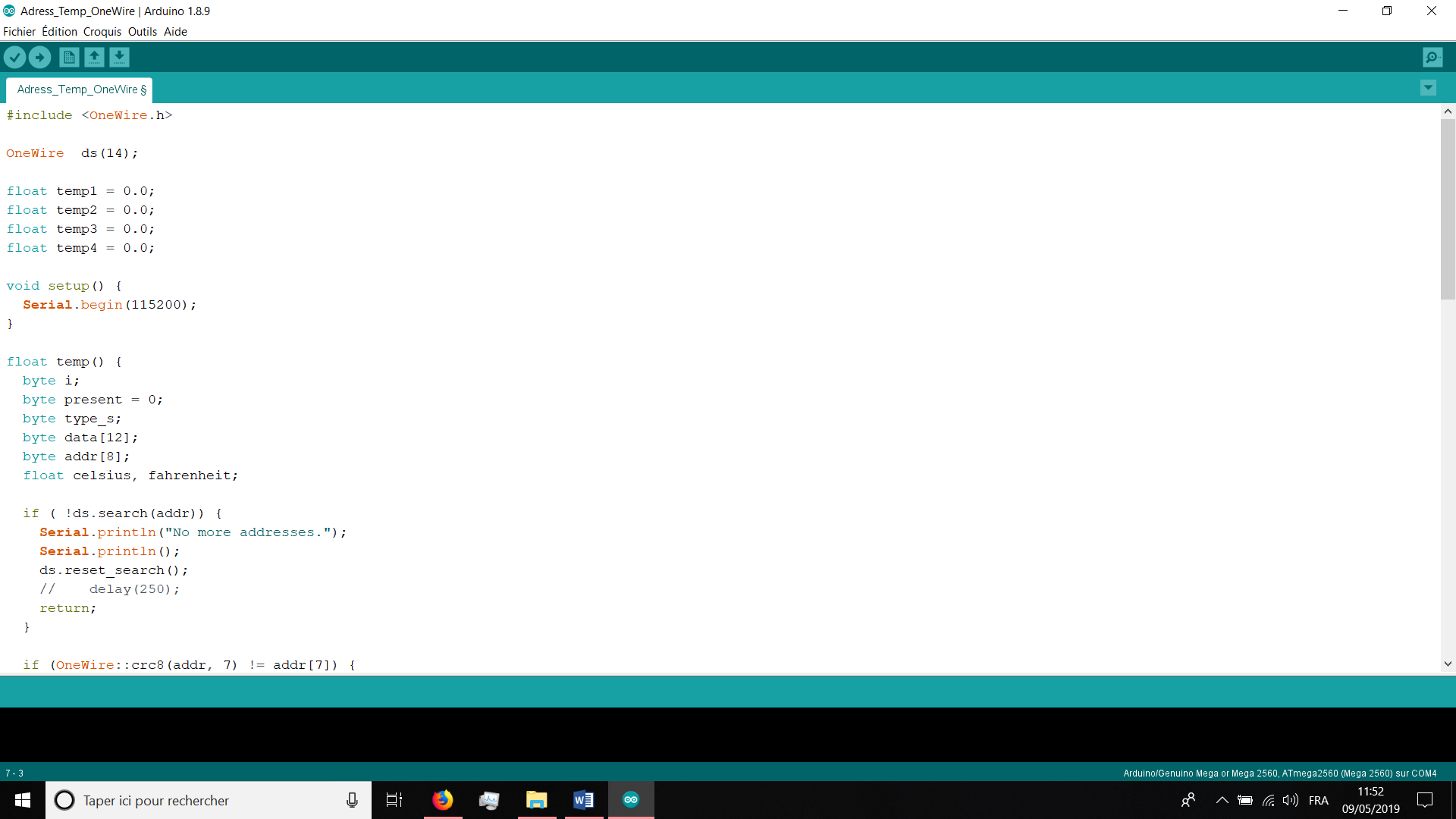
Dans le cadre du projet de la station géothermique, il faut mettre en place ce système sur quatre capteurs différents.

Le programme doit donc jouer en deux temps :

Le premier, situé dans la fonction *temp*, va consister à scanner les adresses des capteurs. Le deuxième, situé dans le *loop,* va récupérer les températures à partir de ces adresses. Pourquoi ? Chaque capteur a une adresse sur 8 octets, toutes les adresses sont donc différentes. Ce système nous permet de récupérer les températures des quatre capteurs sans que l’on ait à connaître leur adresse au préalable. Si jamais un des capteurs dysfonctionne, il suffit juste de le remplacer et de relancer le programme sans devoir retoucher au code.

Vcc GND Data

Une fois l’adresse de chaque capteur récupérée, on peut par la suite obtenir chacun de leur température avec le code suivant, et ainsi pouvoir traiter ces données à notre bon vouloir.



Mise en place des variables et de la pin numérique à utiliser.

La fonction *temp* va récupérer l’adresse d’un capteur et prendre ses données. Les données vont être convertie en une valeur de la température en °C.



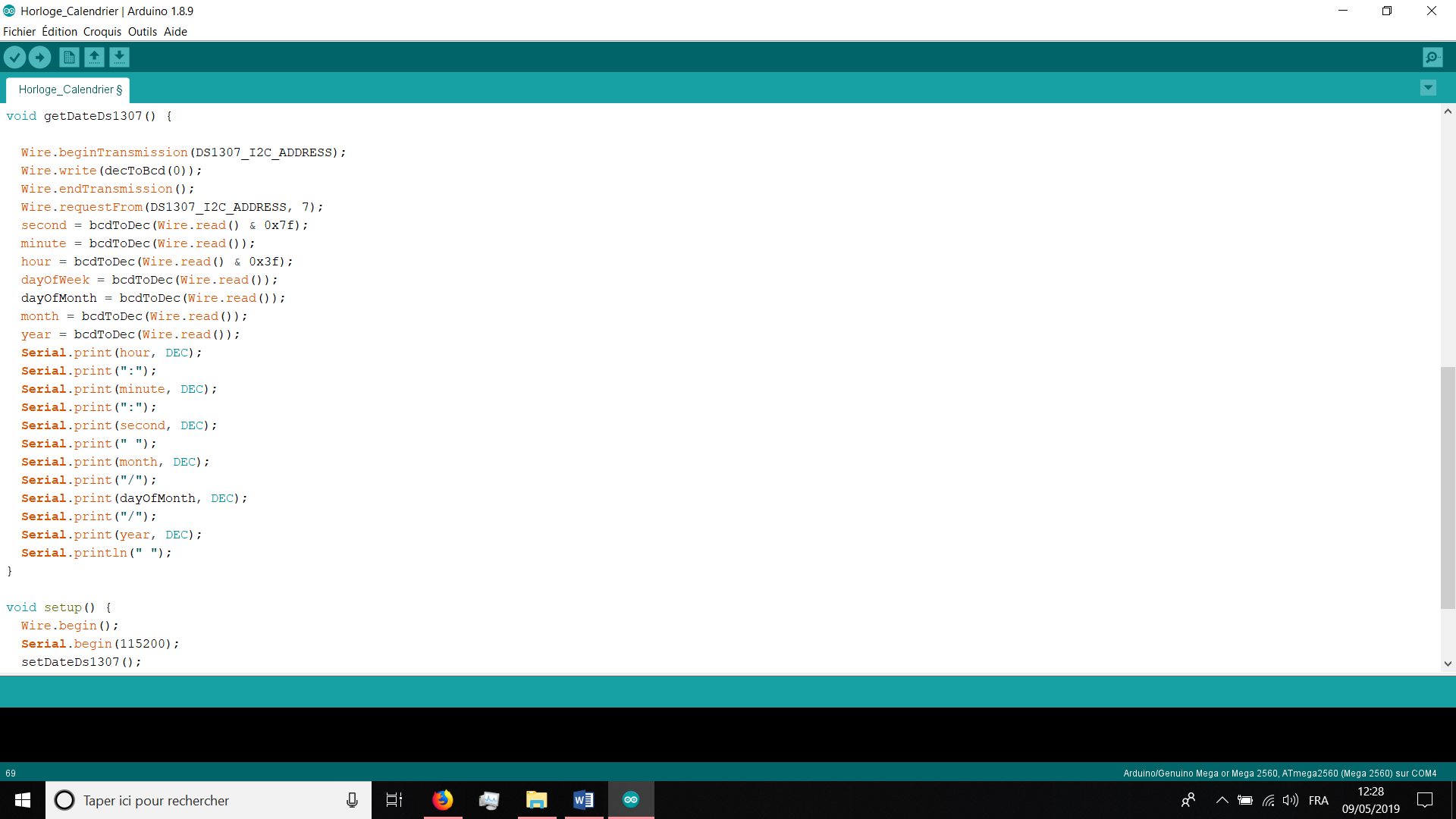
Loop appelant la fonction *temp*, puis l’imprime sur le port série. On pourra par la suite récupérer ces variables pour les envoyer sur le Google Sheet, par ondes radios, et l’imprimer sur la carte SD.

**Gestion de l’horloge :**

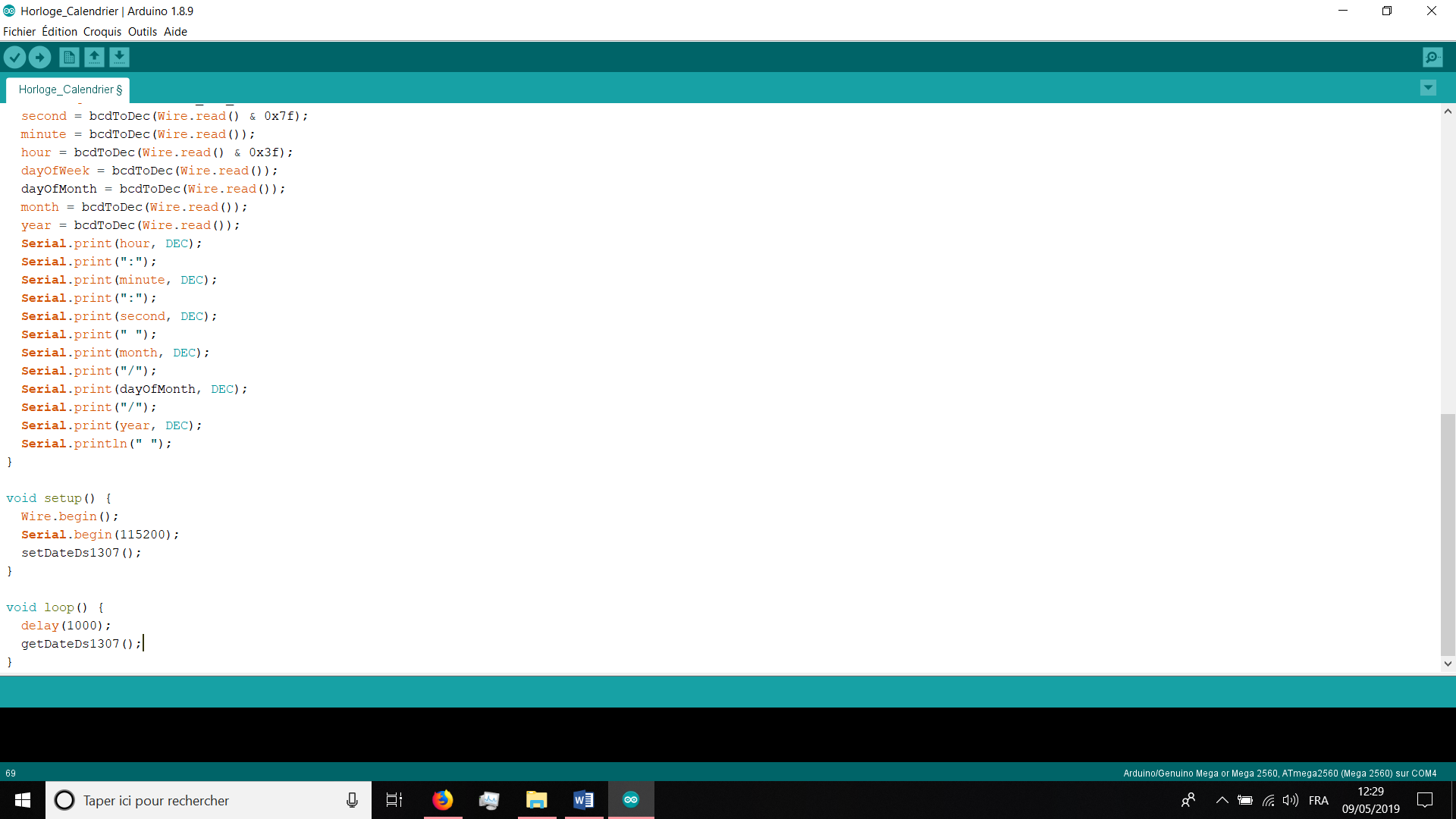
Le microprocesseur Tiny RTC permet de gérer la date et l’heure. Le code se constitue en deux parties bien distinctes, soit une partie de configuration de la date, puis une partie de traitement. Il faut bien faire attention à mettre la date et l’heure correspondantes à chaque fois que l’on lance le programme sur la carte Arduino, si on veut qu’elles soient correctes. Donc devoir ouvrir le code et le modifier.

On peut donc configurer le jour de la semaine, le jour dans le mois, le mois, l’année, l’heure, les minutes et enfin les secondes.





La fonction *getDateDs1307* va faire tourner l’horloge et va permettre de récupérer ces variables pour les imprimer sur le port série ou la carte SD.



La fonction *setup* va appeler la fonction de configuration de la date, puis *loop* va permettre de faire tourner et évoluer la date.



SCL (clock)

SDA (data/données)

VCC (5V)

GND (Ground/masse)

**Transmission sur Google Sheet via Ethernet :**

Les fonction *bcdToDec* et *decToBcd* permettent de faire les conversions décimale/code BCD.

La fonction *setDateDs1307* va permettre de configurer l’heure et la date de départ du programme.

Il s’agit de la partie la plus longue à mettre en place, qui se divise en trois parties :

* Le traitement des données et l’envoi par Ethernet
* La réception sur le site PushingBox, qui va faire la jonction avec un Google Script

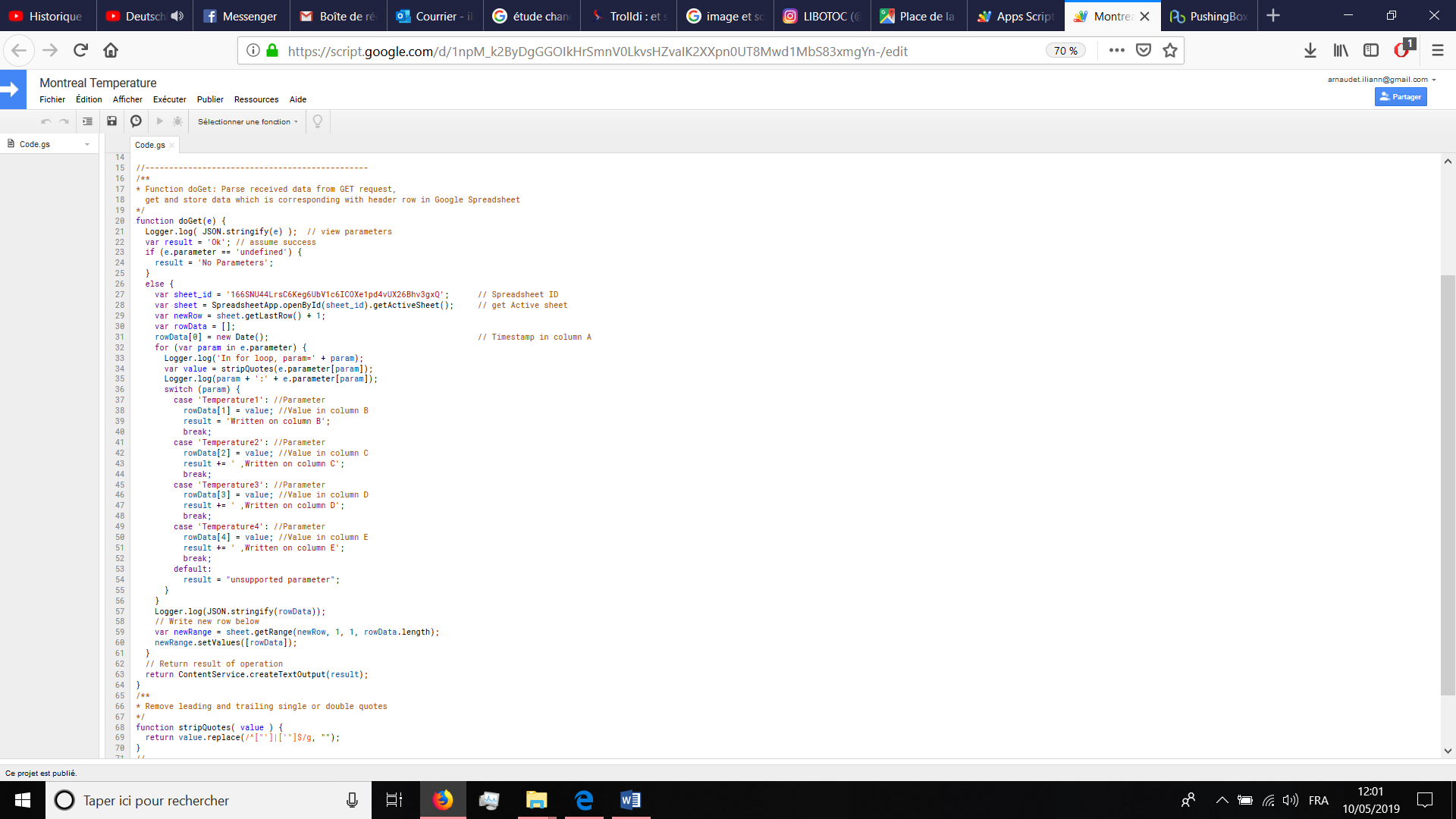
(Arduino ne peut pas directement passer par le Google Script pour des soucis de protocole)

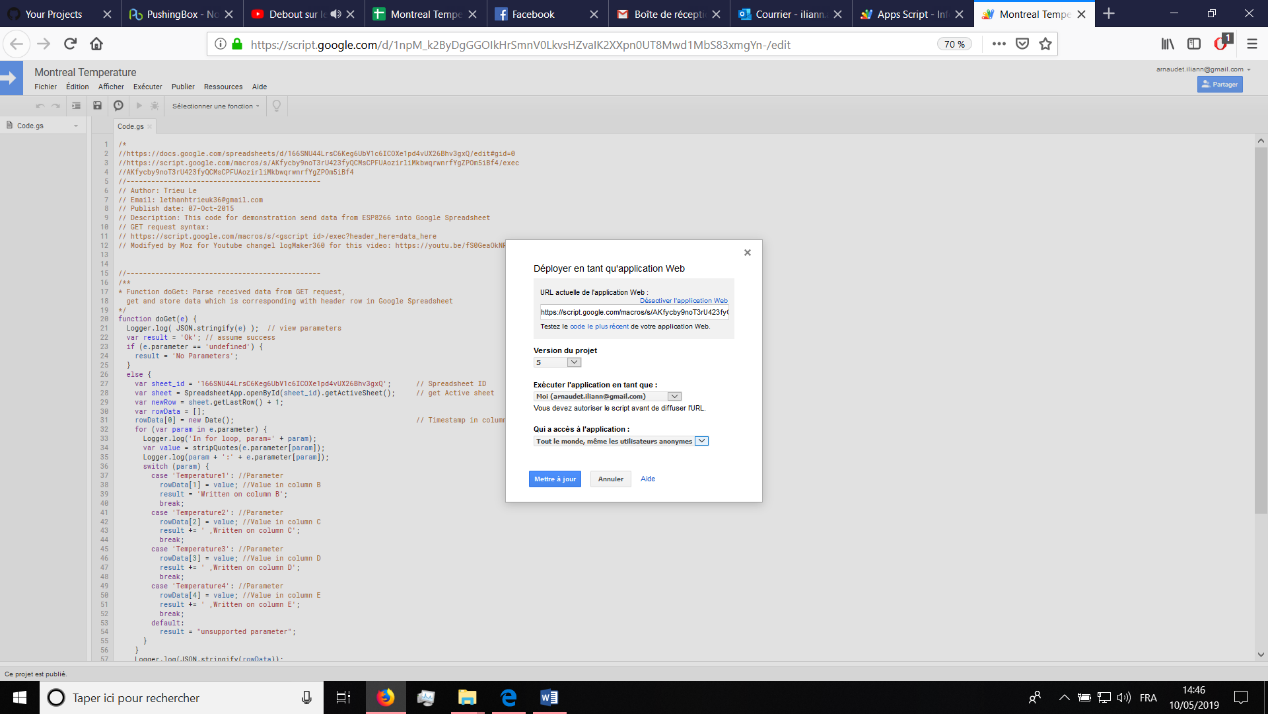
* Le Google Script qui va directement éditer la page Google Sheet correspondante

Pour mettre cela en place, il faut tout d’abord commencer par créer le Google Sheet des données ET son Google Script (applications accessibles à partir du moment où l’on possède un compte Google). Une fois le Sheet créer, il faut copier son ID situé dans son lien web :

*https://docs.google.com/spreadsheets/d/****166SNU44LrsC6Keg6UbV1c6ICOXe1pd4vUX26Bhv3gxQ****/edit#gid=0*

Cette partie

Et le coller sur le code du Google Script ci-présent à cet endroit.

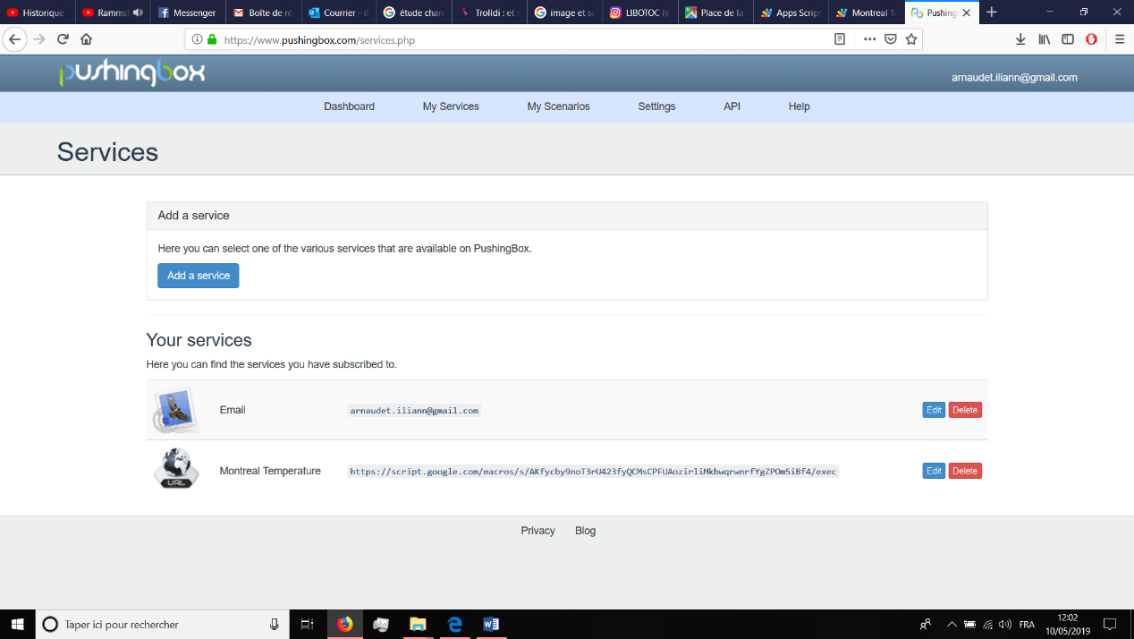
Ensuite, il faut appuyer sur l’onglet *publier,* puis *Déployer en tant qu’application Web*, et ensuite remplir ces différentes parties (sauf l’URL qui est déjà présent). Attention à bien mettre *Moi* et *Tout le monde, même les utilisateurs anonymes*.

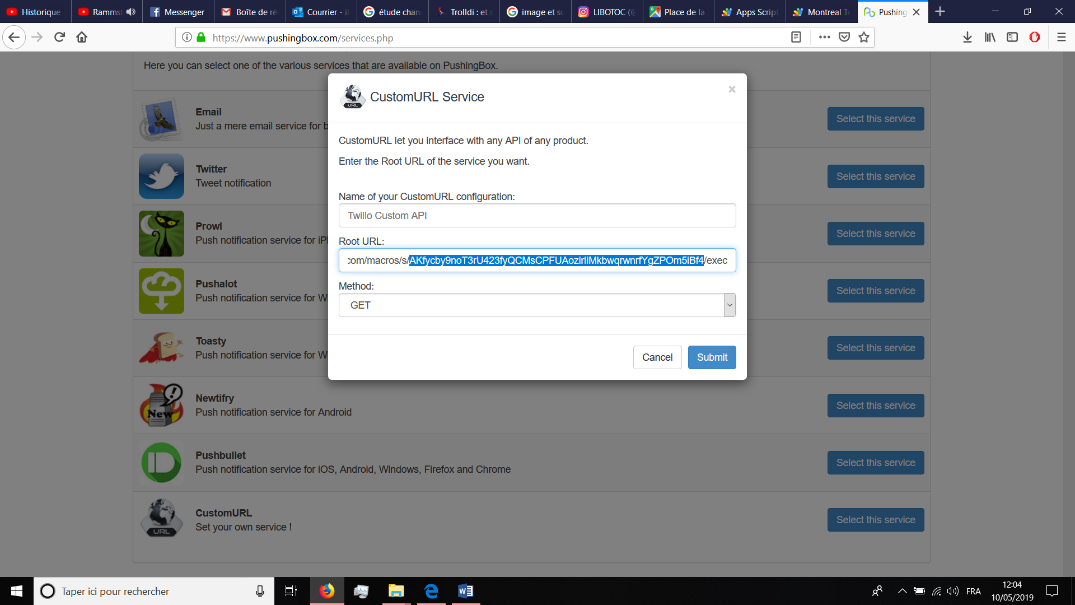
Il faut aussi penser à copier l’adresse du script situé ici :

*https://script.google.com/macros/s/AKfycby9noT3rU423fyQCMsCPFUAozirliMkbwqrwnrfYgZPOm5iBf4/exec*

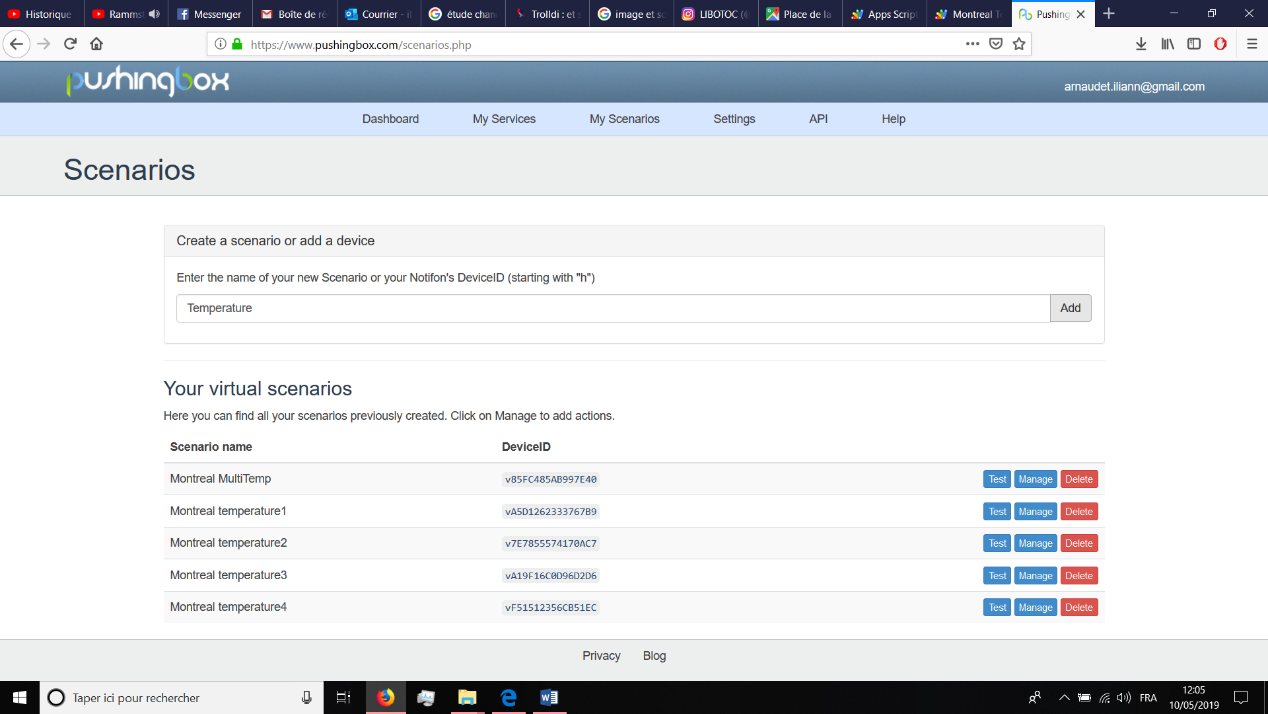
Maintenant, on passe à la partie PushingBox. Il s’agit d’un site qui va faire la jonction entre Arduino et le script. Son utilisation est gratuite, mais limitée à 1000 données échangées par jour (cela nous donne environ une dizaine de mesures par heure, ce qui est largement suffisant pour ce projet). PushingBox utilise les comptes Google.

Pour commencer, il faut aller dans *Service* puis créer son service :





On utilise CustomURL Service, et on colle le lien du script dans Root URL.

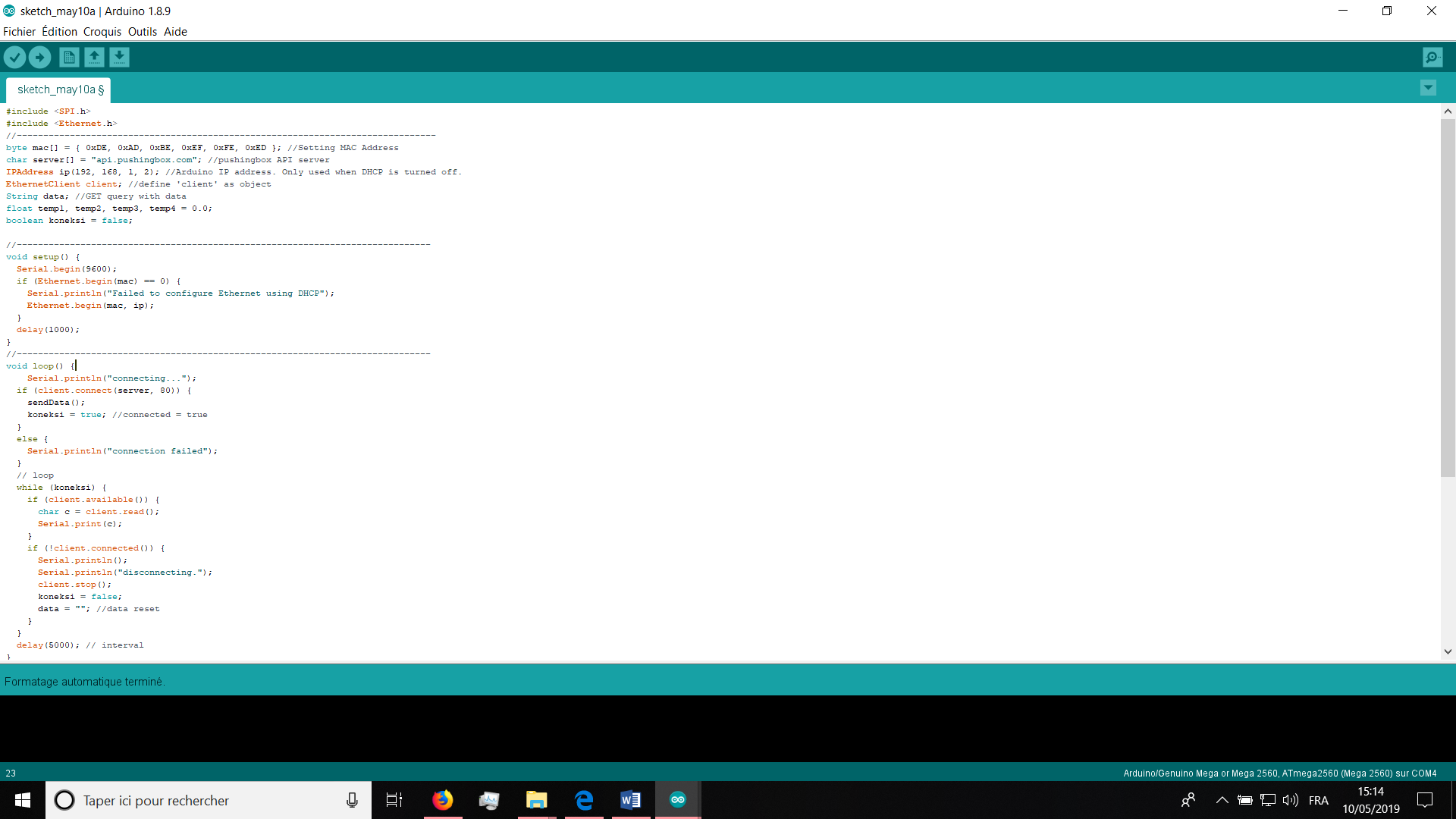
On passe maintenant à la création de *Service*, situé dans le menu du même nom.

On crée un scénario avec n’importe quel nom. On ajoute une action à ce scénario (on prend le Service que l’on vient de créer), et on copie colle la ligne de code suivante : *?Temperature1=$Temperature1$&Temperature2=$Temperature2$&Temperature3=$Temperature3$&Temperature4=$Temperature4$*

* Notons que l’on peut rajouter autant de capteur que l’on veut, il suffit juste de rajouter la mention *TemperatureX=$TemperatureX$&.*

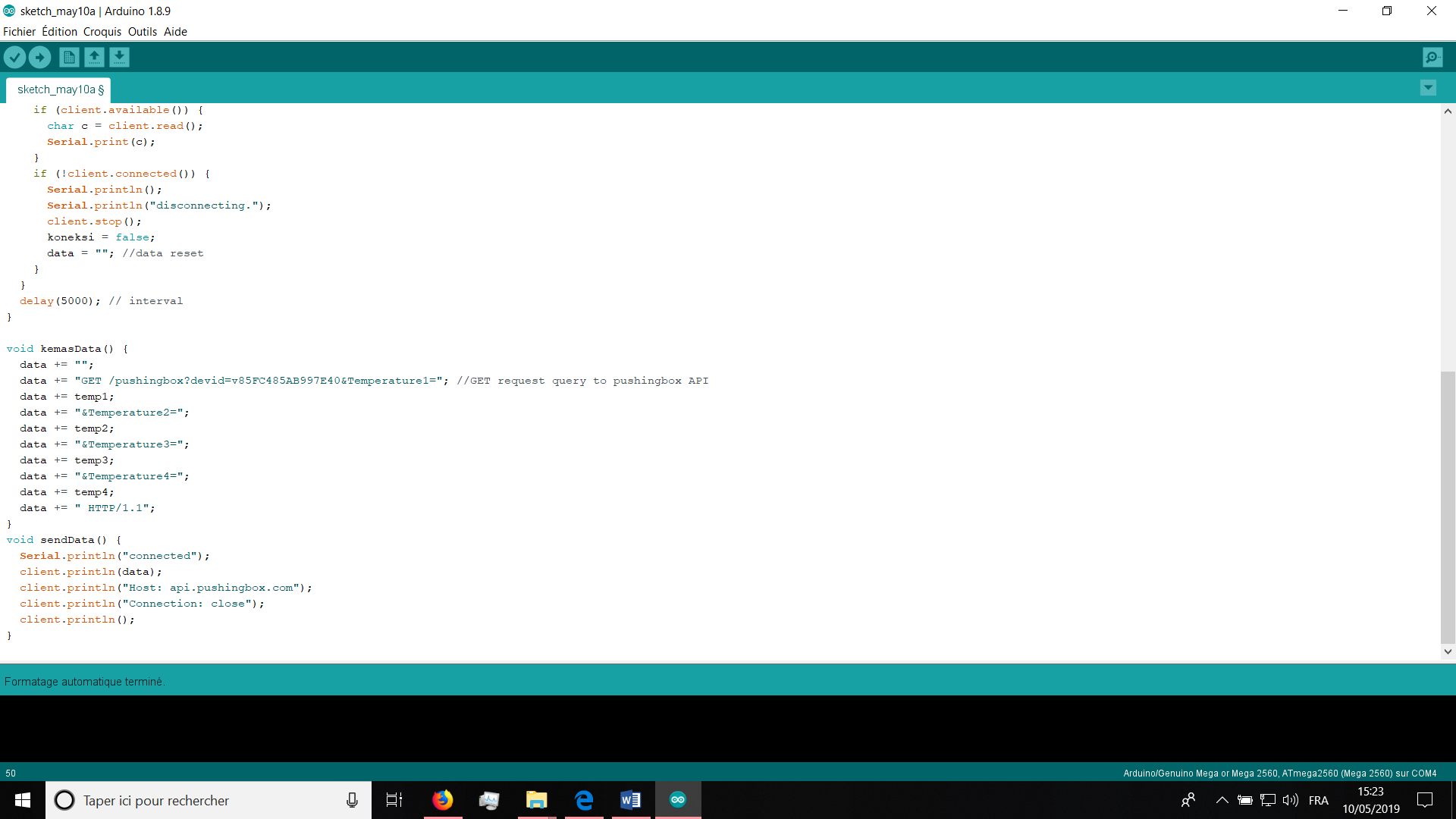
Une fois le scénario terminé, il faut copier son DeviceID qui ressemble à ce genre de suite de caractères : v85FC485AB997E40

Pushing box est maintenant prêt à être utilisé.

Maintenant il ne reste plus qu’à utiliser le programme Arduino

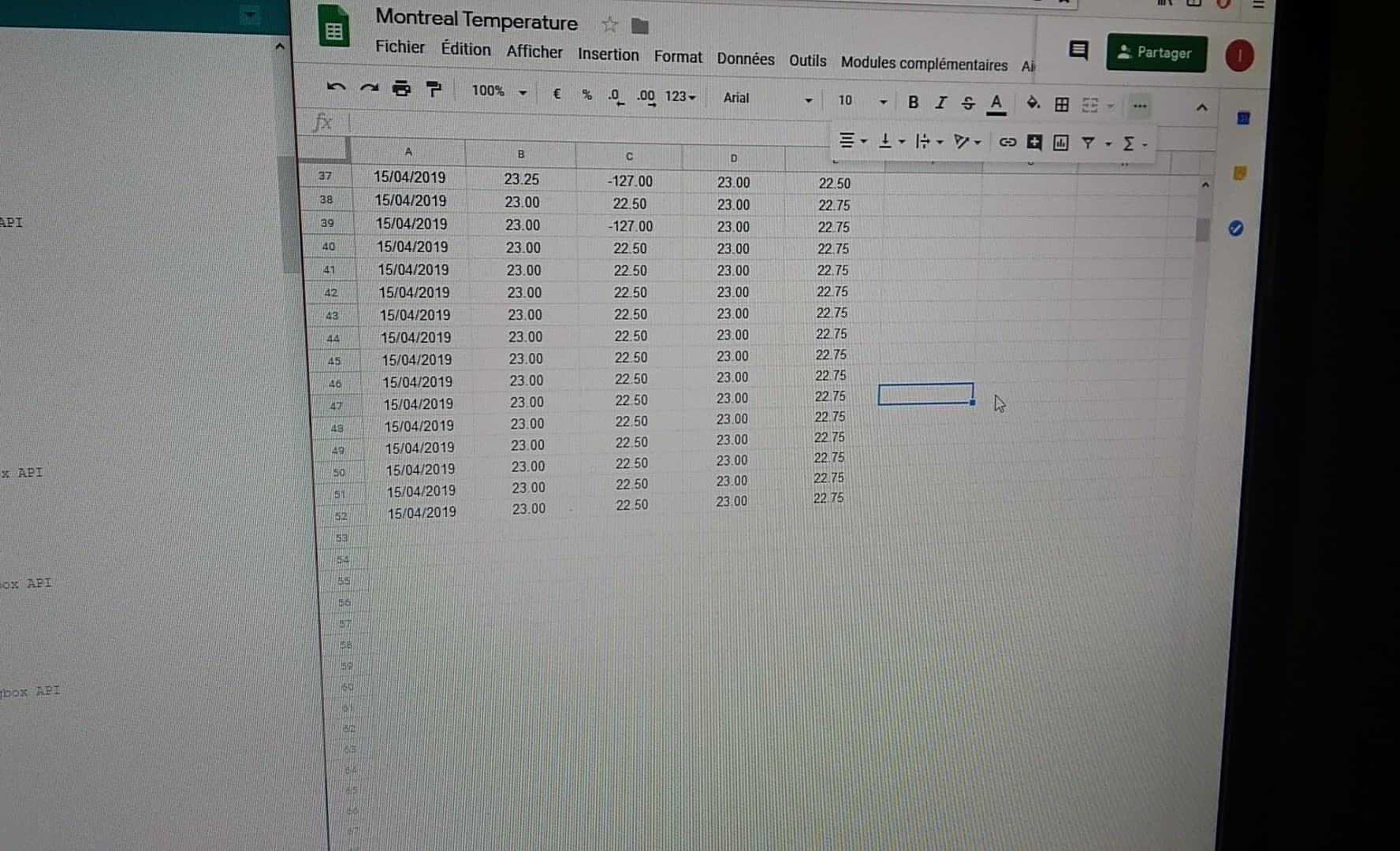
Mise en place des variables, et initialisation des librairies SPI et Ethernet. Test de la liaison Ethernet.

*Loop* qui va faire boucler l’envoi des données en passant par les fonctions *kemasData* et *sendData.*



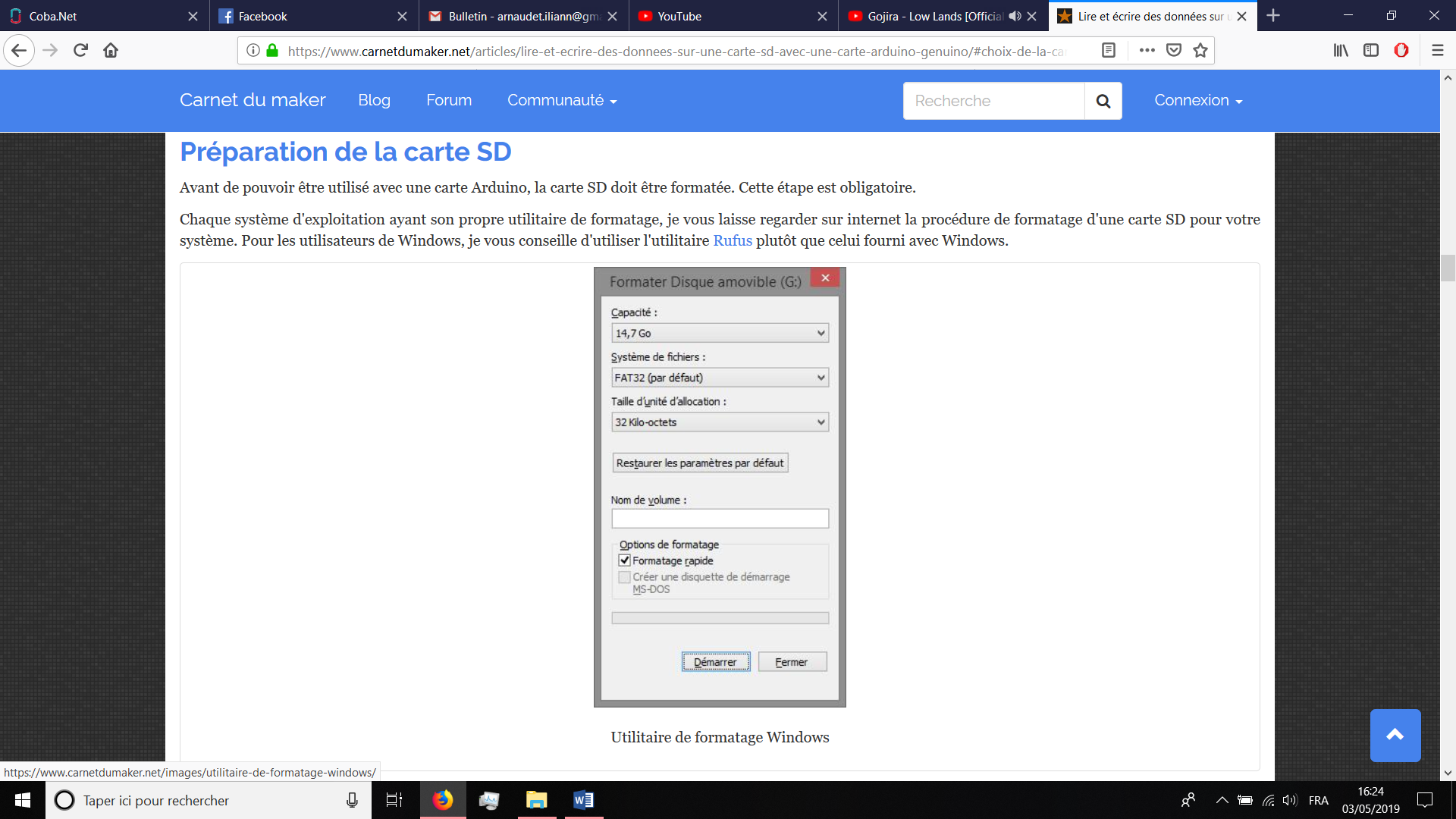
La fonction *kemasData* va préparer l’adresse à envoyer à PushingBox, et *sendData* va l’envoyer.

Une fois toutes ces étapes réalisées, les températures apparaissent comme par magie sur le Google Sheet.



Je renvoi également à l’excellent tutoriel de Arezar272, qui explique parfaitement comment procéder : <https://www.youtube.com/watch?v=fVBqUeksR1I>

**Stockage des données sur carte SD :**

Cette partie est relativement simple, d’autant plus que la librairie à utiliser est déjà compris dans le téléchargement de base du logiciel Arduino. Cela dit, il faut quand même préparer en amont la carte SD en question. Tout d’abord, celle-ci ne devra pas avoir un espace de stockage supérieur à 32GB. Après ça, il va falloir formater la carte SD d’une certaine manière, avec ces paramètres : 

…sans ça, le programme risque de ne pas marcher. Ensuite il faut créer un fichier .txt nommé tout simplement et sobrement *1*. Ensuite il faut créer un fichier nommé *config*, et écrire les lignes suivantes :

frequency = \*nombre de mesures par jours que vous voulez mettre\*

station = \*numéro de la station\*

hour = \*heure correspondante\*

minute = \*minute correspondante\*

second = \*seconde correspondante\*

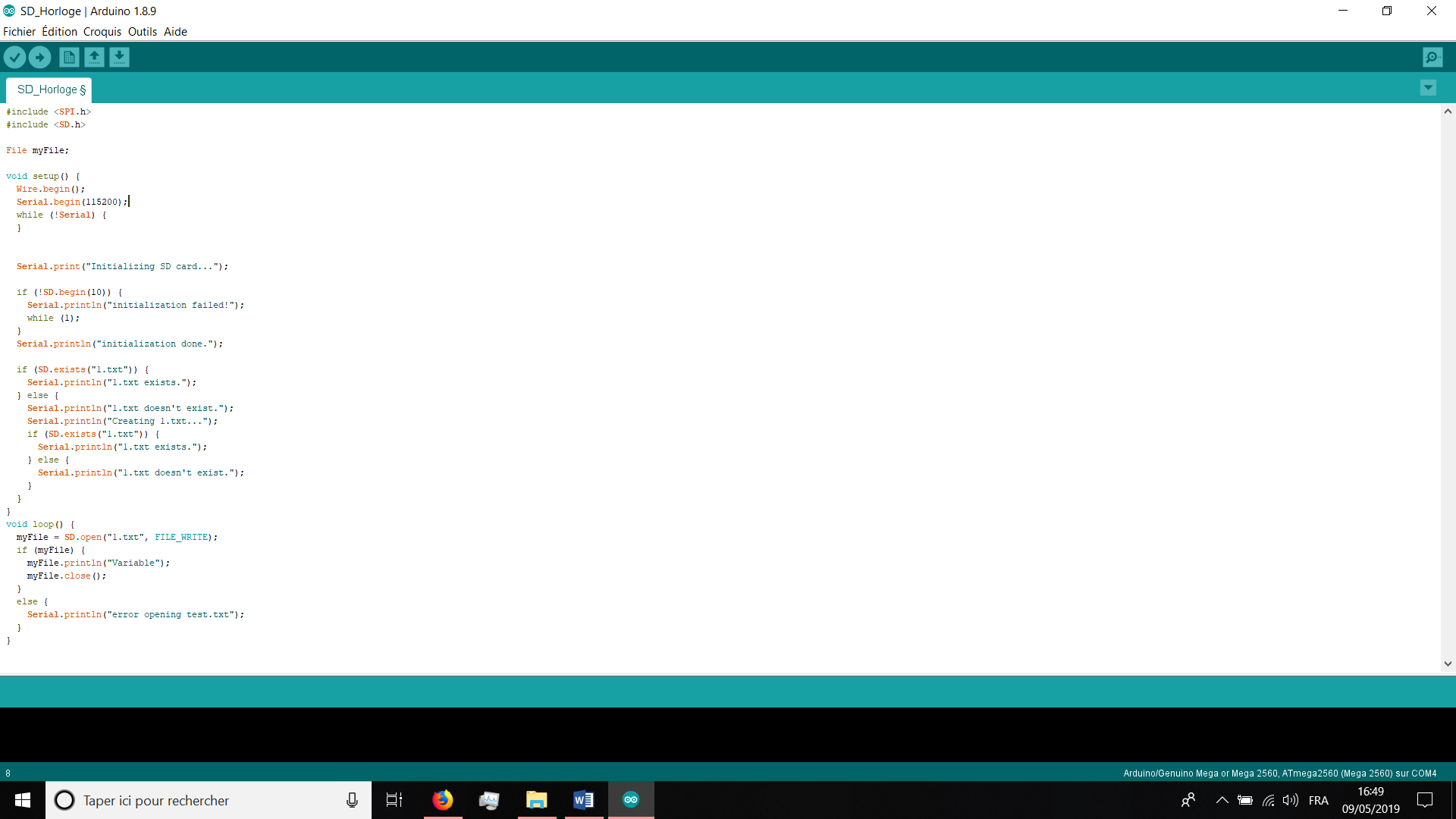
month = \*mois correspondant\*

dayOfMonth = \*jour correspondant\*

year = \*année correspondante\*

Puis il suffit juste d’écrire ces différentes parties de code :

- La première est située dans le *void setup* et permet d’initialiser la carte SD, de savoir si elle permet d’être éditée, ce qu’elle contient...etc.

- La deuxième est située dans la boucle, et va permettre d’ouvrir le dossier correspondant au stockage des données de température sur la carte SD, de l’éditer, et ensuite de le refermer.

Initialisation des paramètres de la librairie SD.

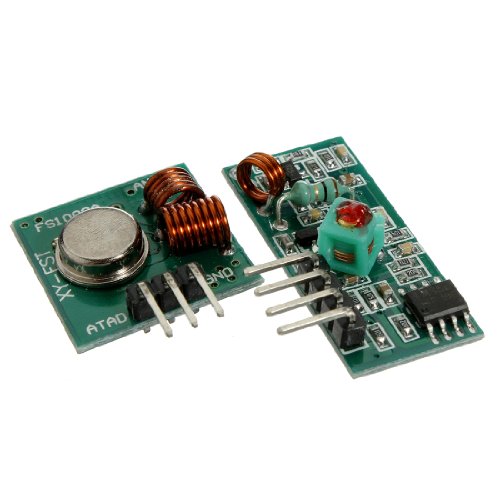
Initialisation de la carte SD, et teste pour savoir si le fichier .txt est accessible.

Partie édition du fichier .txt.

**Communication par ondes radio FM :**

Cette partie communication contient deux codes distincts, le code émission par la carte Arduino Mega 2560 et le code réception par la carte Arduino UNO. Dans ces deux cas il faut mettre en place des antennes de 17cm environ sur chaque module.

L’émission :



*Ground*

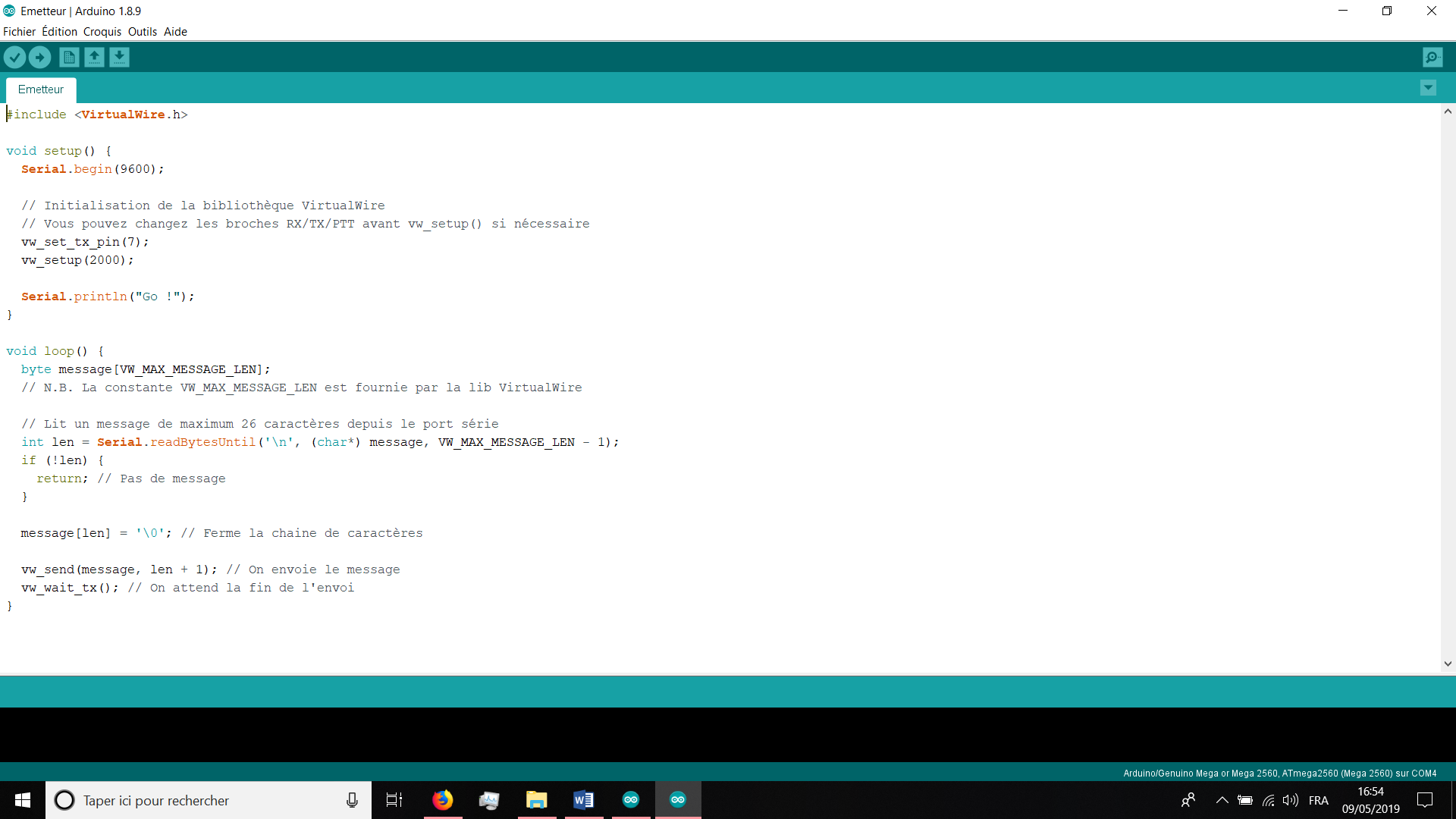
*Data*

*Vcc*

Pour pouvoir émettre les données il faut que l’émetteur ait une antenne. Un câble en cuivre avec une gaine en plastique peut suffire (avec la partie haute du câble dénudée). Elle se soude sur un petit trou du module où il est écrit *Ant*.

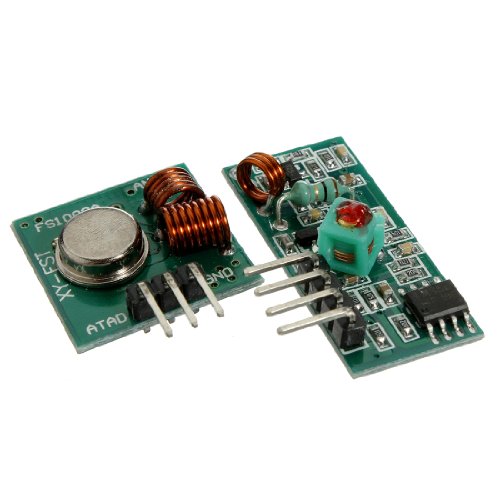
Le programme d’émission se découpe deux parties :

* Une partie d’initialisation dans le *setup*
* Et la partie d’émission des données située dans le *loop,* qui va permettre d’envoyer les données par ondes radios (433MHz en théorie, mais cette donnée n’est pas importante).

Voici le code :

Initialisation de la librairie Virual Wire permettant la communication par radio, ainsi que de la pin émettrice (ici la 7).

Définition de la variable message, et envoi de celui-ci.



La récéption :

*Ground*

*Vcc*

*Data*

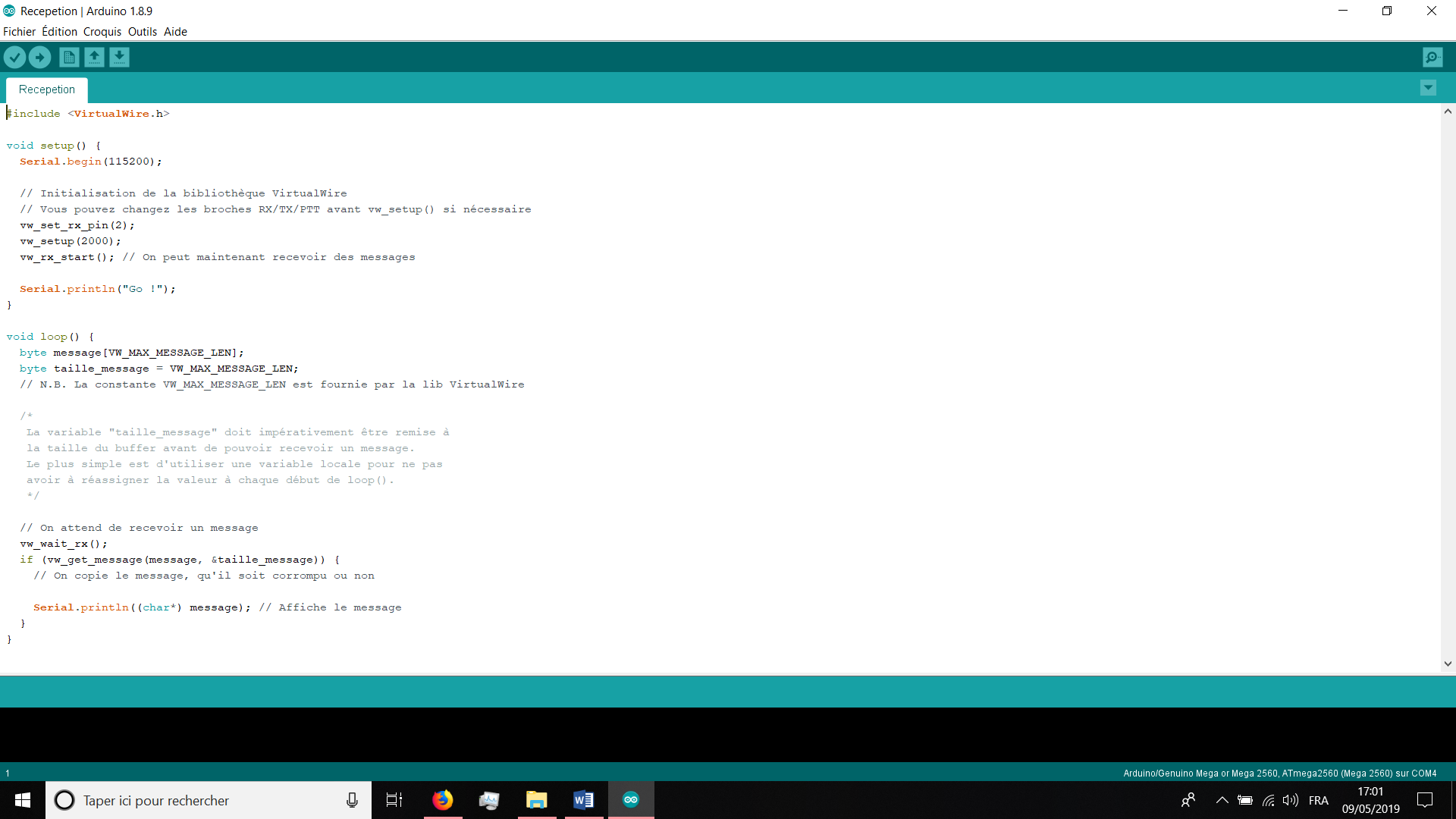
Pour pouvoir recevoir les données il faut que le récepteur ait lui aussi une antenne. Un câble en cuivre avec une gaine en plastique peut suffire (avec la partie haute du câble dénudée). Elle se soude sur un petit trou du module où il est écrit *Ant*.

Le programme d’émission se découpe deux parties :

* Une partie d’initialisation dans le *setup*
* Et la partie de réception située dans le *loop,* qui va permettre de récupérer les données et les définir sur des variables pour ensuite les manipuler à notre guise (comme l’afficher sur le port série ou l’écran LCD).

Voici le code :

*Note :* Ce code est combiné avec le code de l’horloge Tiny Rtc pour pouvoir recevoir les données avec le temps affiché.



Initialisation de la librairie Virual Wire permettant la communication par radio, ainsi que de la pin réceptrice (ici la 2).

Définition de la variable message, et réception de celui-ci.

**Affichage sur écran LCD :**

Cette partie fait partie des plus simples, le code est relativement court ainsi que simple à comprendre. Les seules erreurs qui peuvent être commises sont au niveau du câblage.

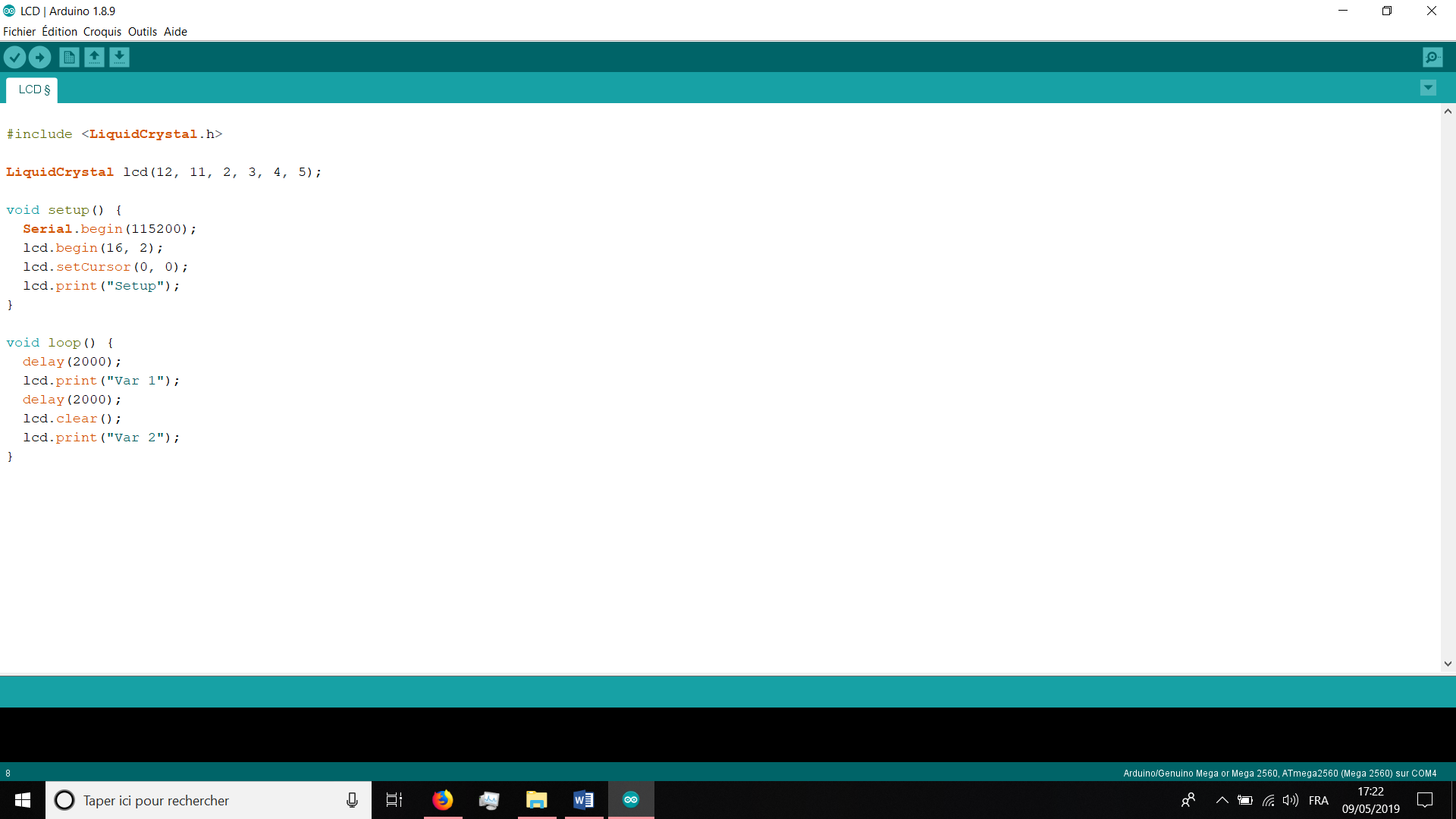
Notes : L’écran LCD n’est volontairement pas mis avec la carte arduino mega 2560, pour une bonne raison. L’écran utilise énormément de pins spécifiques, et cela bloque l’utilisation de la carte SD. Il est donc préférable (si on veut quand même utiliser l’écran LCD) de le brancher sur la carte UNO, qui sert de réception des ondes radios.

Le programme est divisé en deux parties :

- une initialisation de l’écran située dans le *setup*

- la partie affichage située dans la boucle *loop*

On notera que 16,2 signifie que l’on peut mettre 16 caractères sur une ligne, et qu’il y’en a deux (ces paramètres changent en fonction de l’écran utilisé, en l’occurrence un 16,2 fait largement l’affaire pour ce projet).



Initialisation de la librairie LiquidCrystal, essentielle au bon fonctionnement de l’écran LCD.

Initialisation des pins de la carte arduino UNO qui vont être utilisées.

Initialisation du curseur.

Ecriture sur l’écran LCD.

**Interface et gestion de la station**

Nous pouvons accéder à la visualisation du traitement des données en utilisant le port série. Sinon, d’un point de vue configuration, il est aussi possible de régler la station sur certains paramètres en accédant à la carte SD de celle-ci. Sur la carte SD se trouve un dossier *config* avec différents paramètres réglables.

Voici toutes les configurations rendues disponibles :

* Fréquence (explicitée en nombre de mesures par jour)
* Numéro de la station
* Le temps (c’est-à-dire l’heure et la date, configurable à la seconde près)