

## COMPTE RENDU PROJET



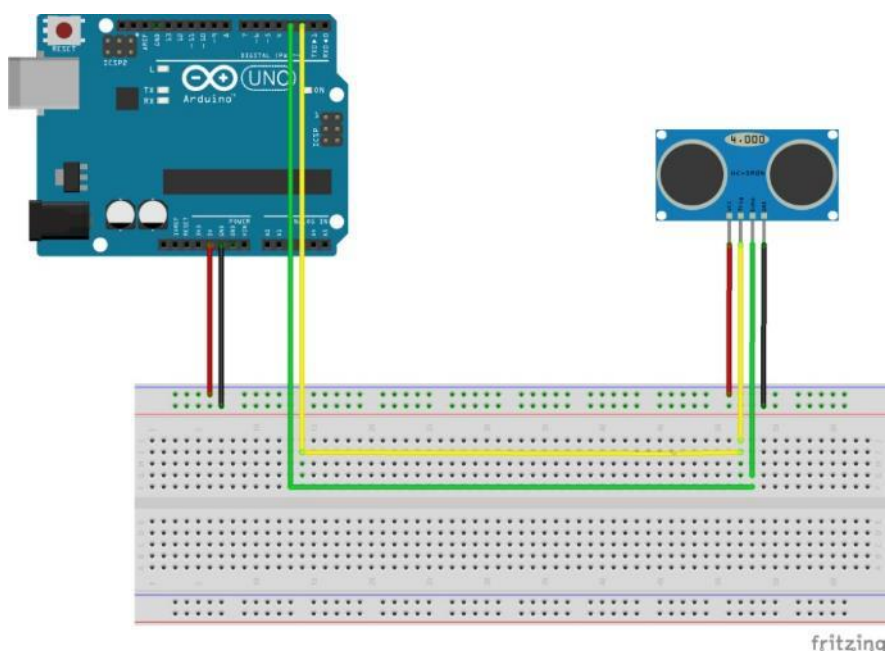
# Table des matières

Table des matières .....	2
Présentation .....	3
Base de Linux.....	4
Les bases de Linux .....	5
Les capteurs.....	6
Arduino .....	7
Raspberry .....	10
Programmation du HAT du Raspberry Pi 3.....	11
Le capteur de présence .....	14
Le Capteur ultrason .....	17
Communication Arduino-Raspberry.....	19
Bouton et LED.....	20
La photo résistance .....	23
Soudage .....	25
Le thermocouple .....	26
Ecran LCD 16*2.....	27
Node Red .....	29
Objectifs futurs.....	29
Fiche Pratique.....	30



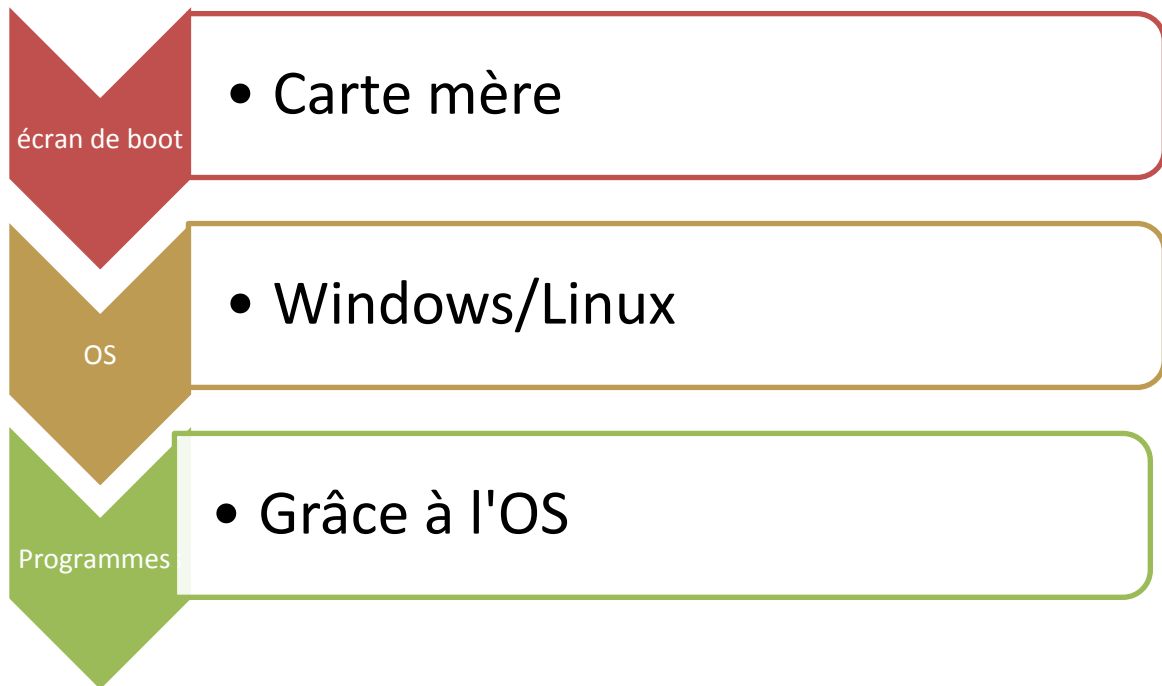
# Présentation

Ayant pour but de créer un projet, je dois d'abord mettre en œuvre plusieurs sortes de capteurs pour pouvoir les réutiliser plus tard. L'objectif est donc de débiter la programmation avec Python, et d'avoir les bases de Linux. Ensuite il faudra s'intéresser aux capteurs. Ces capteurs fonctionneront grâce à deux cartes qui sont Arduino et Raspberry que nous verrons également. Les programmes que j'ai mis dans ce compte rendu ont tous été testés. J'ai également souhaité mettre plus de détail que pas assez pour permettre à toute personne débutante de comprendre ce que je voulais effectuer.



# Base de Linux

Linux est un système d'exploitation (OS : Operating System) qui fonctionne comme Windows. Lorsque l'on démarre son ordinateur, il y a un écran de boot qui apparaît avec le fonctionnement de la carte mère, ensuite on décide de L'OS qui va continuer à faire fonctionner l'ordinateur donc soit Windows, soit Linux (ou autre comme Mac OS...). On peut enfin lancer les programmes grâce à L'OS.



Il faut noter qu'il est possible d'installer Windows et Linux sur le même ordinateur et de choisir son OS au démarrage (écran de boot).

L'avantage de Linux est qu'il y a beaucoup plus de logiciel gratuit, et ils sont plus souvent mis à jour. Il est en plus possible de faire fonctionner les logiciels Windows grâce à Wine. Certains logiciels Linux sont plus performants que ceux sur Windows. Un avantage intéressant est que les logiciels sont libres, ils sont dits en "open source". Chacun peut avoir le code et le modifier comme il le souhaite. Du fait des codes libres il existe plusieurs distributions de Linux (Mandriva, Red Hat, Debian, Slackware) et il existe encore des dérivées des distributions (par exemple pour Debian : Kaella, Knoppix, Ubuntu, Skolelinux).

Il est possible d'installer une distribution de Linux sur une carte SD et de la faire fonctionner avec une Raspberry. Sur Linux il faut programmer grâce à une invite de commande. Nous verrons plus tard comment faire pour installer L'IDE Arduino par exemple (IDE:integrateddevelopmentenvironment en anglais, environnement de développement en français).



# Les bases de Linux

Un PDF sur internet<sup>1</sup> assez complet permet de regrouper beaucoup d'éléments pour la programmation Python. Dans ce PDF on peut retrouver :

- Les types et opérations de bases : nombre/ booléens/ chaîne de caractères/ liste.
- Les structures de contrôle : if/ elif/ else/ for/ while.
- Les fonctions.
- Les fichiers.
- Les classes.

Il est possible de faire des petits tests pour vérifier si l'on a bien compris l'utilité des fonctions

Voici un petit programme réalisé sur Python sur une carte Raspberry avec un HAT (explication dans la partie Raspberry.)



Figure 1 : logo Linux

```
fromsense_hat import SenseHat      : important la bibliothèque du SenseHat
sense = SenseHat()
```

```
sense.show_message("Hello"         : message à écrire
                      , scroll_speed=0.1 : vitesse de défilement sur le panneau de LED
                      , text_colour=[255,255,0] : couleur du texte qui défile
                      , back_colour=[0,0,255] ) : couleur de l'arrière-plan
```

Il est possible de faire une multitude de programmes avec Python, des petits programmes comme ceux-ci, ou bien sûr de bien plus gros. Certains logiciels facilitent la programmation comme Node-RED par exemple.

---

<sup>1</sup>[http://calcul.math.cnrs.fr/Documents/Ecoles/2010/les\\_bases.pdf](http://calcul.math.cnrs.fr/Documents/Ecoles/2010/les_bases.pdf)



# Les capteurs

Un capteur est un dispositif transformant l'état d'une grandeur physique observée en une grandeur utilisable, telle qu'une tension électrique, une hauteur de mercure, une intensité ou la déviation d'une aiguille.

On peut retrouver une grande liste de capteurs sur le site internet GOTRONIC<sup>2</sup> compatible avec les cartes Arduino. Un autre site<sup>3</sup> propose également une liste de capteurs correctement répertoriés.

En voici une petite liste : capteurs ultrason, infrarouges, pression, RGB, UV, capteur de gaz, de pluie , de température, accéléromètre, capteur piezzo, capteur PIR, interrupteur tilt, cellule de charge, détecteur à effet hall, transducteur ultrason....

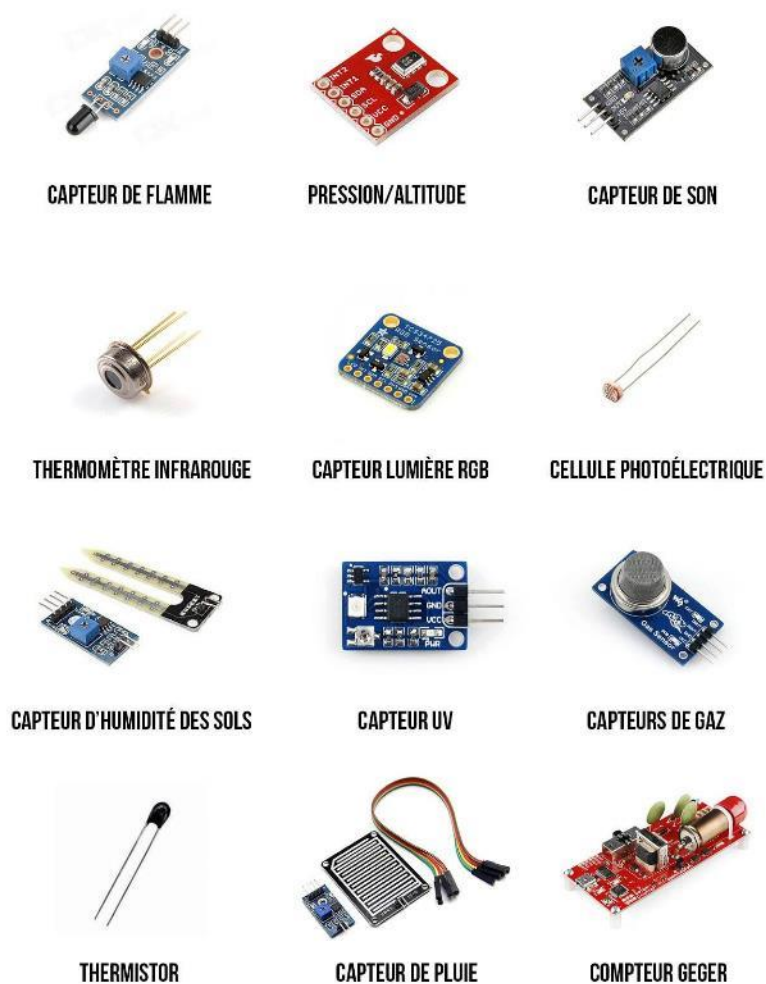


Figure 2 : plusieurs capteurs disponible sur gotronic...

<sup>2</sup><http://www.gotronic.fr/cat-capteurs-1114.htm>

<sup>3</sup><http://www.lafabriquediy.com/tutoriel/liste-des-capteurs-229/>



# Arduino

Arduino est une marque Italienne de cartes libres sur lesquelles se trouve un microcontrôleur. Les schémas de ces cartes sont publiés en licence libre mais certains composants, comme le microcontrôleur, ne sont pas sous licence libre. Cette carte a pour objectif d'analyser et de produire des signaux électriques sur les broches, de manière à effectuer des tâches très diverses comme la domotique (le contrôle des appareils domestiques - éclairage, chauffage...), le pilotage d'un robot, de l'informatique embarquée... La carte est basée sur une interface entrée/sortie simple. Elle peut être utilisée pour construire des prototypages rapides ou peut être utilisée pour des projets importants. Il existe une multitude de capteur, de moyen de communications (écran LCD, Wi-Fi, Bluetooth, Gsm...) compatible avec les cartes Arduino. Le codage est un C++, ce qui permet d'obtenir des bases concrètes très rapidement. De plus il est possible de retrouver sur internet un grand nombre de protocole/tutoriel pour effectuer différentes tâches ou pour mettre en œuvre un grand nombre de capteur (ou autre..).



Voici une carte Arduino UNO, assez basique qui permet d'effectuer déjà beaucoup de chose. Pour des projets importants, le nombre de broches sera trop faible, il faudrait plutôt choisir une Arduino méga. La carte UNO coûte 20€ sur le site officiel Arduino (<https://www.arduino.cc/>) tandis que la carte MEGA coûte 35€.

Figure 3 : carte Arduino

Les cartes Arduino ont déjà été utilisées dans de nombreux projets importants, comme des robots aspirateurs, ou bien d'autres robots encore. Elle est en effet très pratique car adaptable à énormément de projet. Voici un exemple de robot :

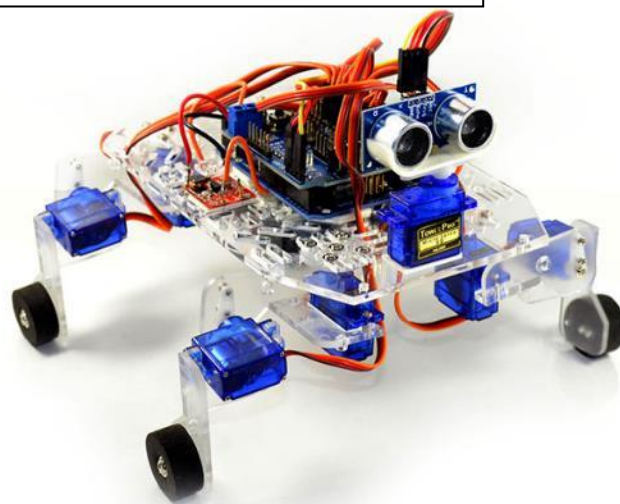


Figure 4 : Robot avec une Arduino



Pour présenter les montages fait sur une Arduino, il est possible d'utiliser le logiciel Fritzing (voir : <http://fritzing.org/home/> ). Grâce aux nombreux composants qu'il propose, il est possible de recréer son montage réel en montage virtuel beaucoup plus propre qu'une photographie. Il est même possible de voir le circuit imprimé du montage... Voici un exemple de montage Arduino réalisé par mes soins sur Fritzing :

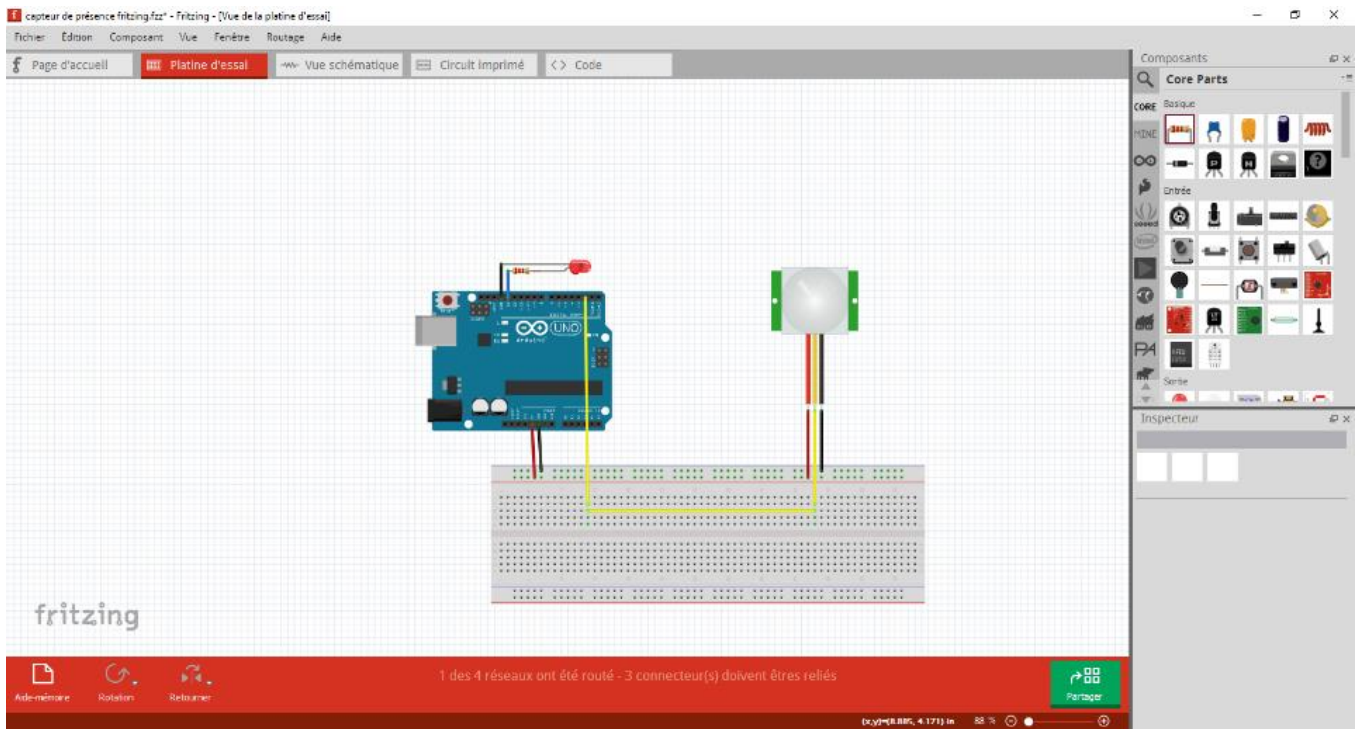


Figure 5 : montage Fritzing

Et voici le montage réel, beaucoup moins facile à comprendre. Il est donc très utile de faire ses montages sur Fritzing pour pouvoir réutiliser les anciens montages et comprendre son fonctionnement :

Remarque : Le montage présenté ici est un capteur de présence. Lorsque que le capteur capte un mouvement, il le transmet à la carte, qui nous avertit d'un mouvement en allumant la LED rouge. Il est possible bien sûr d'imaginer que la LED peut être une lampe, lorsque quelqu'un passe dans un couloir ou autre...

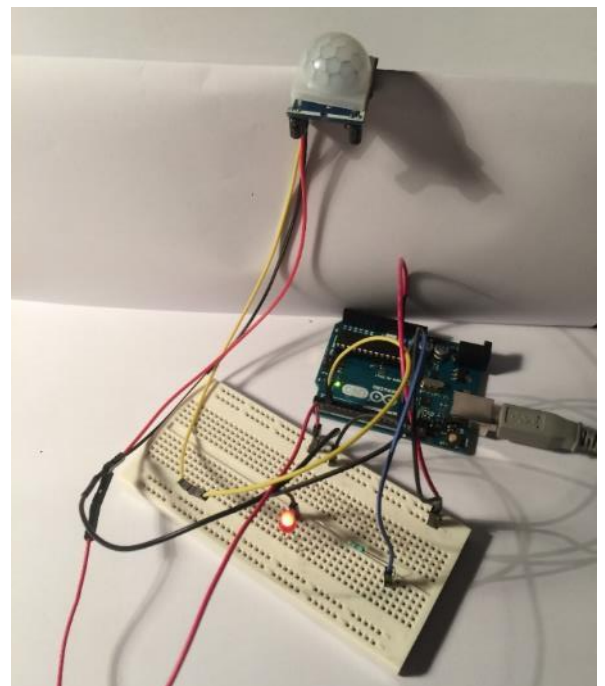


Figure 6 : montage réel

Ensuite le code Arduino est assez simple à comprendre et s'apprend assez vite. En voici une partie, pour le codage du montage ci-dessus :

```
int calibrationTime = 10; // temps que l'on accorde au capteur de présence pour s'initialiser ( entre 10 et

int ledPin = 13;           // broche de la LED
int inputPin = 2;          // Entrée de la valeur du capteur
int pirState = LOW;        // we start, assuming no motion detected
int val = 0;               // valeur de la lecture du capteur

void setup() {
  pinMode(ledPin, OUTPUT); // la broche de la LED est une sortie
  pinMode(inputPin, INPUT); // la broche du capteur est une entrée
  Serial.begin(9600);

  Serial.print("calibration du capteur ");
  for(int i = 0; i < calibrationTime; i++){
    Serial.print("."); // mettre des petits points, un point par seconde jusqu'à la fin de la calibration
    delay(1000);
  }
}

void loop(){
  val = digitalRead(inputPin); // lire la valeur du capteur et la stocker dans val
  Serial.println(val);         // écrire la valeur val
  if (val == HIGH) {           // Si la valeur = 1 ( high ) alors il y a un mouvement
    digitalWrite(ledPin, HIGH); // allumer la LED
    delay(150);

    if (pirState == LOW) {
      Serial.println("mouvement detecte!");
    }
  }
}
```

Voici une rapide présentation. Premièrement il faut déclarer toutes les variables à utiliser dans le programme. Il faut également déclarer les variables des broches utilisées. Par exemple, `int ledPin = 13` ; veut dire que lorsque j'écrirais `digitalWrite(ledPin, HIGH)` ; il y aura la mise sous tension de la broche associée à `ledPin`, soit la numéro 13. Après avoir défini les variables, il faut faire l'initialisation, cela se fait dans le `void setup()` { INITIALISATION }, il faut définir si les broches sont des entrées sorties, allumer le moniteur série, et bien d'autre chose comme ici lancer la calibration du capteur. Enfin le `void loop()` { BOUCLE } est fait pour faire tourner en boucle le programme que l'on conçoit, c'est la partie active du code.

Enfin dernier point, le moniteur série, il apparaît sous forme d'une fenêtre, et il est utile pour communiquer avec l'utilisateur. On peut lui demander de nous transmettre des mesures, ou de nous afficher un message si une action est effectuée, par exemple dans le programme, il est écrit que s'il y a un mouvement, alors la carte transmet le message «mouvement détecté» au moniteur série. Voici un des bases de programmation du moniteur série :

`Serial.begin(9600)` ; allumer le moniteur série en 9600 bauds

`Serial.print(« texte : »,valeur)` ; écrire un message texte : valeur

`Serial.println(« texte : »,valeur)` ; même chose que précédemment mais on écrit à la ligne

Remarque :

- « `//` » est utilisé pour faire des commentaires
- Le baud est une unité de mesure utilisée dans le domaine des télécommunications en général, et dans le domaine informatique en particulier. Le baud est l'unité de mesure du nombre de symboles transmissibles par seconde.
- Le moniteur série peut être utilisé pour déboguer un programme.



# Raspberry

Le Raspberry Pi est tout comme un ordinateur mais en bien plus petit. Elle permet à beaucoup de se lancer dans l'informatique pour un coup peu excessif. Il permet l'exécution de plusieurs variantes du système d'exploitation libre GNU/Linux et des logiciels compatibles. Il est fourni nu (carte mère seule, sans boîtier, alimentation, clavier, souris, écran) dans l'objectif de diminuer les coûts. Tout comme l'Arduino elle permet une mise en place rapide de beaucoup de programme. Elle est compatible avec beaucoup de capteur. Sur une carte SD nous avons installé un Linux avec un certain nombre de programme.

Ce petit ordinateur est bien sur compatible avec l'Arduino. Il ne faut surtout pas les opposer mais les assembler. En effet l'Arduino est extrêmement pratique dans la mise en œuvre de capteur, mais pour communiquer avec l'utilisateur elle se révèle moins bonne. Alors que la Raspberry peut communiquer beaucoup plus facilement avec Node-Red par exemple (voir : <https://nodered.org/> ). Pour un projet il est donc très utile de travailler avec Arduino pour les fonctions mesure, et le Raspberry pour la communication ou le stockage de données.

La petite carte Raspberry pi 3 se présente comme ceci :

Il faut y connecter un écran (port HDMI), un clavier et une souris (port USB). Il faut également avoir Linux sur la micro carte SD et voilà, l'ordinateur est prêt à être exploité.



Figure 7 : carte Raspberry

De plus il existe un HAT (chapeau en français) qui permet un bon nombre de manipulation. En effet elle possède un panneau de LED permettant de faire passer des messages, un joystick...

Il est en plus possible sur le programmeur d'avoir des exemples déjà réalisés de nombreux programmes, très utiles car ils sont fiables et il est donc possible de les exploiter pleinement.



Figure 8 : Hat du Raspberry



# Programmation du HAT du Raspberry Pi 3

Le Hat vu précédemment peut être programmé pour afficher de nombreux messages. Je vais alors y consacrer une partie afin de pouvoir le ré-exploiter si besoin dans la suite de mon projet.

- Afficher un message:

```
fromsense_hat import SenseHat
sense = SenseHat()
sense.show_message("Hello", scroll_speed=0.1, text_colour=[255,255,0], back_colour=[0,0,255] )
```

- Faire une boucle infinie :

```
while True:

    sense.show.message(...)
```

- Pour faire une pause :

```
import time # à mettre tout au début.

sense.show.message

time.sleep(1) #temps en seconde
```

- Pour nettoyer l'écran :

```
sense.clear()
```

- Afficher une seule lettre :

```
sense.show_letter ("lettre", text_colour=[255,255,0], back_colour=[0,0,255] )
```

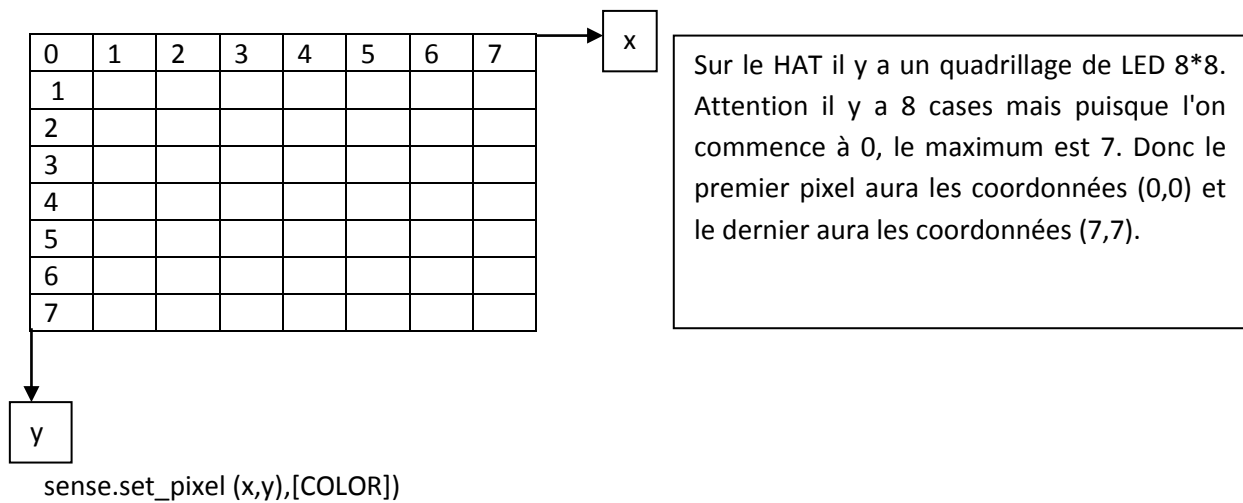
- Nombre aléatoire :

```
import random # à mettre au début

r = random.randint(a,b) # r est un entier aléatoire entre a et b
```



- Afficher des pixels :



- Se définir une couleur :

r= [0,255,255]

Remarque : le codage est un codage RGB (Red,Green,Blue), donc le premier chiffre définit l'intensité du Red (rouge), de 0 (pas de couleur) à 255 (couleur totale), le deuxième chiffre pour le Green (vert), et le dernier pour le blue (bleu). A noter que les chiffres sont codés sur 256 bits (de 0 à 255). Bien sûr en combinant les trois couleurs il est possible d'avoir toutes les couleurs possible puisque le vert le rouge et le bleu sont les couleurs primaires.

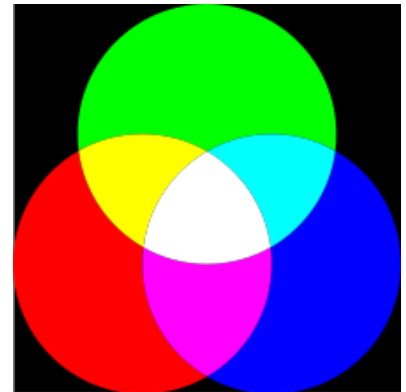


Figure 9 : couleur primaire et associations

- Matrice d'image :

```
image=[
r,r,r,r,r,r,r,r,
b,b,b,b,b,b,b,b,
r,r,r,r,r,r,r,r,
r,r,r,...
r,r,...
r,...
.... ]
sense.set_pixels(image)
```

Le but est de recréer l'image en définissant la couleur de chaque pixel par une lettre préalablement définie comme ci-dessus (se définir une couleur).



Il existe également une chose très intéressante sur le HAT de la Raspberry, c'est plusieurs capteurs tels qu'accéléromètre, température, pression, humidité, gyroscope...

```
from sense_hat import SenseHat # librairie du HAT
import time # permet de mettre des pauses

sense = SenseHat()
sense.clear() # nettoyage de l'écran

while True : # permet de faire une boucle infinie

    temp = sense.get_temperature() # on lit la temperature
    pres = sense.get_pressure()# on lit la pression
    humi = sense.get_humidity()# on lit l'humidité

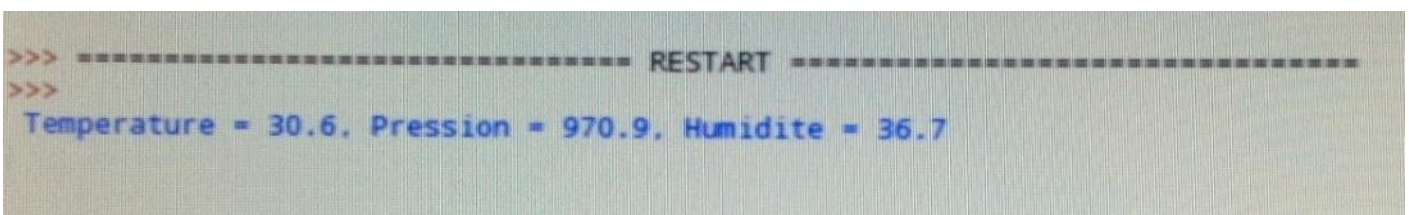
    temp = round ( temp, 1 ) # on arrondit à un chiffre après la virgule
    pres = round ( pres, 1 )
    humi = round ( humi, 1 )

    msg = " Temperature = %s, Pression = %s, Humidite = %s" % (temp,pres,humi)
    sense.show_message(msg,scroll_speed = 0.07) # on écrit sur l'écran de led du HAT
    print(msg) # on écrit sur le moniteur série

    time.sleep(2) # attendre 2 secondes
```

Sur l'écran on peut voir défiler le texte du moniteur série.

Et voici le résultat sur le moniteur série, toute les 2 secondes une nouvelles mesure est effectuée :



```
>>> ===== RESTART =====
>>> Temperature = 30.6, Pression = 970.9, Humidite = 36.7
```

Figure 10 : retour sur le moniteur série



# Le capteur de présence

Le capteur de présence s'appelle également PIR motion (Pyroelectric InfraRed sensor). Il fonctionne facilement, il détecte une variation d'onde infrarouge et génère alors un courant.

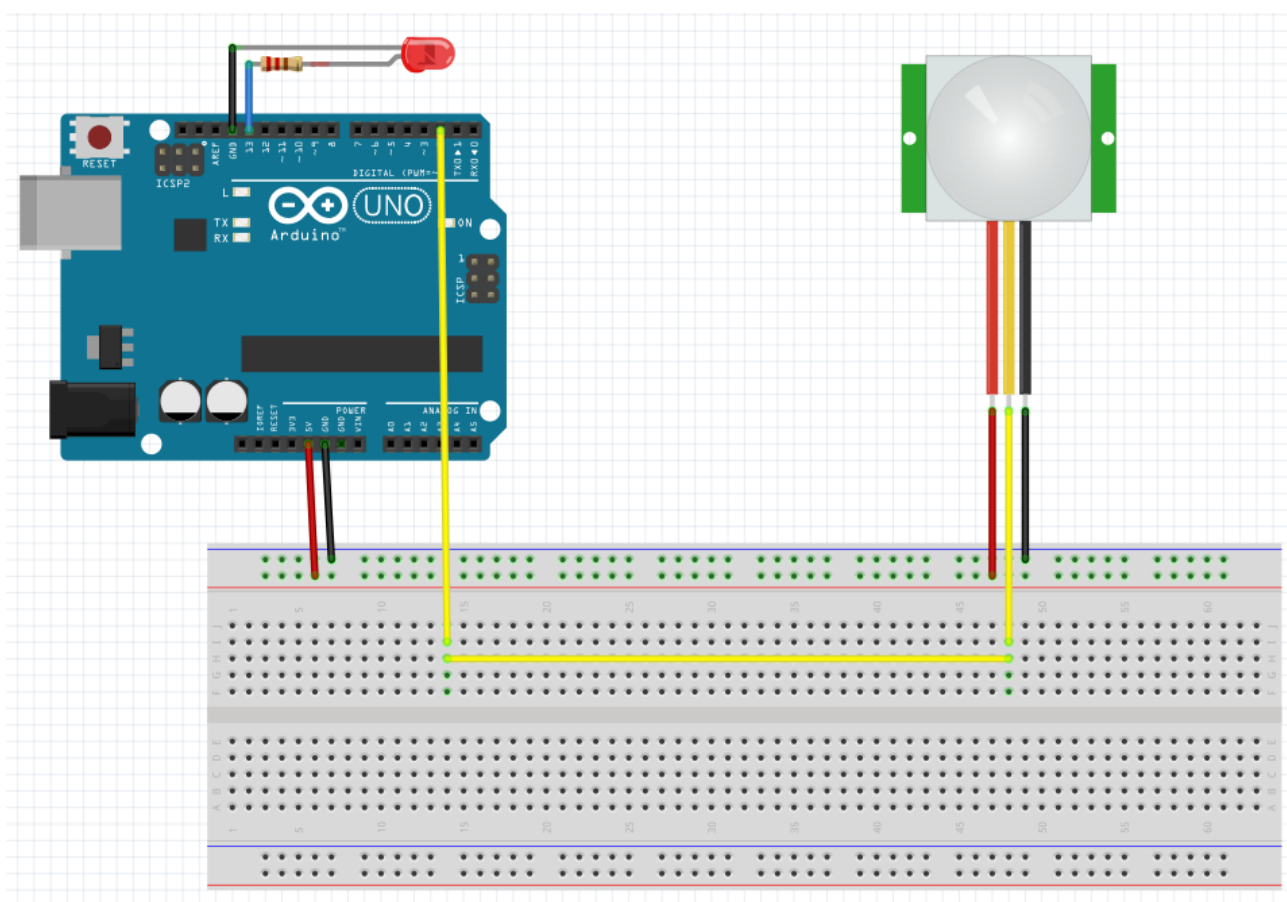
L'objectif sera de:

Lire la valeur du PIR (Vrai : mouvement détecté; Faux : aucun mouvement).

Transmettre cette valeur à l'utilisateur.

Allumer une LED et transmettre des messages à l'utilisateur.

Il faut faire le montage (sur Fritzing)



Pour ce montage assez simple, il faut :

Une Arduino UNO, une résistance 300 Ohm, une LED (rouge ici), une breakboard, et des fils.



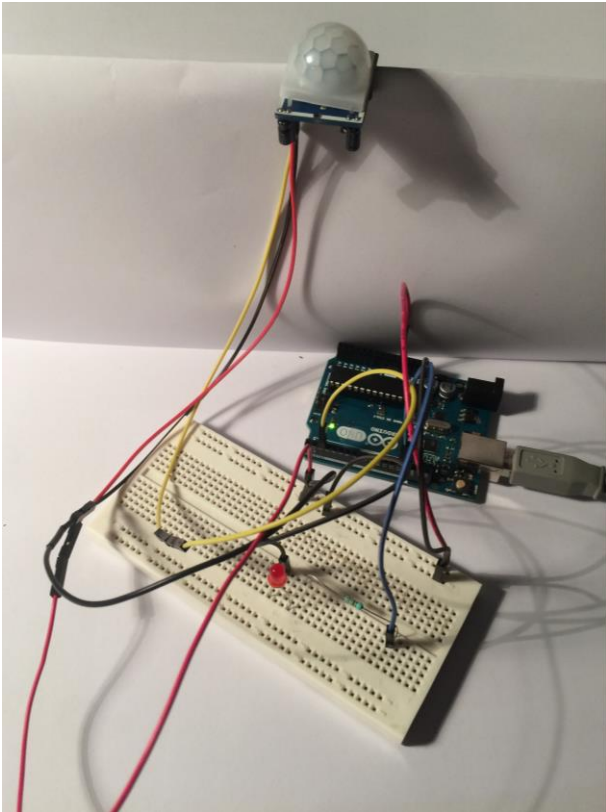


Figure 11 : montage réel

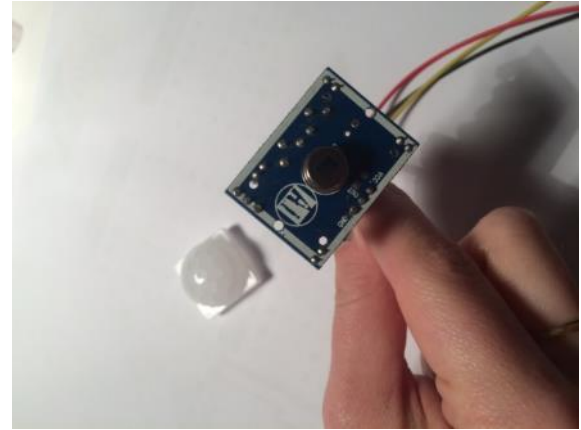


Figure 12 : capteur PIR

Le capteur PIR se cache sous une petite capsule blanche.

A chaque fois que l'on passe devant le PIR, une LED s'allume et un message est envoyé sur le terminale Arduino en disant la valeur du capteur (0 ou 1) et lorsqu'il passe à 1 il envoie un message comme quoi un mouvement est détecté, et lorsqu'il passe à 0 il envoie un message comme quoi il y a un mouvement. De plus au départ il y a la calibration donc il est transmis sur la terminale qu'il y a la calibration qui est effectuée avec une succession de petit point le temps de la calibration.

Il y a des commentaires (//commentaires) à côtés des lignes pour préciser à quoi elles correspondent.

Pour plus de précision sur les LED, j'y consacrerai une partie rapide dans bouton+LED



Le code pour le capteur de présence est le suivant :

```
int calibrationTime = 10; // temps que l'on accorde au capteur de présence
pour s'initialiser (entre 10 et 60 secondes)

int ledPin = 13;           // broche de la LED
int inputPin = 2;          // Entrée de la valeur du capteur
int pirState = LOW;        // on commence avec aucun mouvement détecté
int val = 0;               // valeur de la lecture du capteur

void setup() {
  pinMode(ledPin, OUTPUT); // la broche de la LED est une sortie
  pinMode(inputPin, INPUT); // la broche du capteur est une entrée
  Serial.begin(9600);

  Serial.print("calibration du capteur ");
  for(int i = 0; i < calibrationTime; i++){
    Serial.print("."); // mettre des petits points, un point par seconde
    jusqu'à la fin de la calibration
    delay(1000);
  }
}

void loop(){
  val = digitalRead(inputPin); // lire la valeur du capteur et la stocker
  dans val
  Serial.println(val);         // écrire la valeur val
  if (val == HIGH) {           // Si la valeur = 1 ( high ) alors il y a
  un mouvement
    digitalWrite(ledPin, HIGH); // allumer la LED
    delay(150);

    if (pirState == LOW) {
      Serial.println("mouvement detecte!");
      pirState = HIGH;
    }
  }
  else {
    digitalWrite(ledPin, LOW); // éteindre la LED
    delay(300);
    if (pirState == HIGH){
      Serial.println("aucun mouvement");
      pirState = LOW;
    }
  }
}
```



# Le Capteur ultrason

L'objectif sera de :

Mesure la distance entre le capteur et un objet

Transmettre cette distance à l'utilisateur par le biais du moniteur série

Le capteur ultrason fonctionne très simplement : il envoie un ultrason et mesure le temps d'écho si il y'en a un. Sachant que  $v = \frac{d}{t}$  alors  $d = v * t$  ; Or la vitesse du son dans l'air est égale à 340 m/s. Donc en connaissant le temps (que le capteur mesure) on connaît la distance de l'objet qui a réfléchi l'onde.

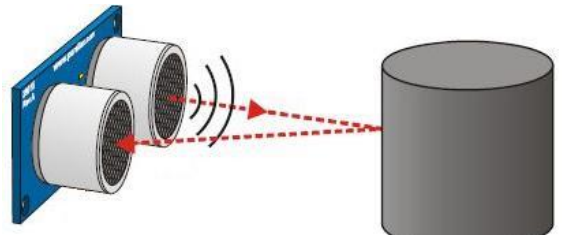


Figure 13 : détection avec les ultrasons

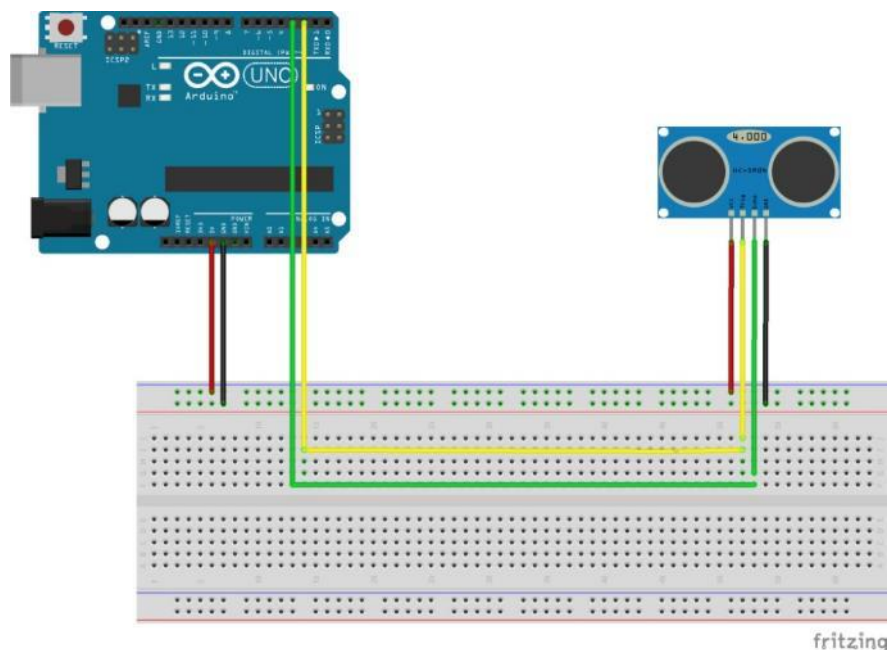
Sur le capteur ultrason (à droite) on peut observer 4 broches. Le vcc (5V), le Gnd (Ground), le Trig (Trigger) et l'Echo. On met état haut de 10 microsecondes sur la broche trig. On remet à l'état bas la broche Trig. On lit la durée d'état haut sur la broche Echo. Puis on divise cette durée par deux pour n'avoir qu'un trajet Enfin on calcule la distance avec la formule  $d = v * t$ .



Figure 14 : capteur à ultrasons

Remarque : pour résultats plus précis, il sera important de prendre en compte la vitesse du son dans l'air en fonction de la température qui a son influence.

Ensuite il faut réaliser le montage:



Pour le code c'est le suivant :



```

#define VITESSE 340 //on définit la vitesse du son à 340 m/s

const int trig = 8; // Trig sur la broche 8
const int echo = 9; // Echo sur la broche 9

void setup() {
    pinMode(trig, OUTPUT); //la broche Trig est une sortie
    pinMode(echo, INPUT); // la broche Echo est une entrée
    digitalWrite(trig, LOW); // on initialise le trig sur basse
    Serial.begin(9600); // moniteur série 9600 bauds
}

void loop()
{
    digitalWrite(trig, HIGH); //état haut du trig
    delayMicroseconds(10); //on attend 10 µs
    digitalWrite(trig, LOW); // état bas du trig

    unsigned long duree = pulseIn(echo, HIGH); // on le temps de retour de
    l'écho

    if(duree > 30000) // si la durée est supérieur à 30 ms
    {
        Serial.println("Onde perdue, mesure échouée !"); // on écrit à la
        ligne que l'onde est perdue et la mesure a échouée
    }
    else // sinon
    {
        duree = duree/2; //divise la durée par 2 pour n'avoir qu'un trajet de
        l'onde
        float temps = duree/1000000.0; //on met en secondes
        float distance = temps*VITESSE; //on multiplie par la vitesse, d=t*v
        Serial.print("Distance = ");
        Serial.print(distance); //affiche la distance mesurée (en mètres)
    }
    delay(500); // pause afin d'éviter d'avoir trop de résultat.
}

```



# Communication Arduino-Raspberry

Sachant que l'Arduino et le Raspberry sont complémentaires, il est important de savoir faire passer des messages de l'une à l'autre.

Pour cela il faut d'abord installer la librairie pyserial pour faciliter la tâche des conceptions de script. Il faut donc connecter le Raspberry à internet et entrer la ligne de commande suivante :

```
apt-getinstall python-serial
```

Tout d'abord nous allons étudier la partie où l'Arduino va envoyer un message au Raspberry et lui va le récupérer :

Code Arduino :

```
intcompteur=0;

voidsetup() {
    Serial.begin(9600);
}
voidloop() {

    Serial.print("Message numero ");
    Serial.println(compteur);
    Serial.println("Communication OK");

    compteur++;

    delay(5000);
}
```

Ici le code sert à envoyer un message toutes les 5 secondes en disant combien de messages il a déjà envoyés (message numéro x) et disant que la communication est OK. On remarque que c'est comme si l'on envoyait un message au moniteur série, d'ailleurs il est également envoyé au moniteur série, c'est simplement le Raspberry qui récupère le message envoyé par la carte Arduino.

Code Raspberry :

```
importserial

ser=serial.Serial('/dev/ttyACM0',9600)

while1:

    print(ser.readline())
```

Le code sert ici à lire ce que l'Arduino envoie.

La fonction "serial.Serial" nous permet de lire les messages reçus. Le premier paramètre de cette fonction est le nom du port usb occupé par l'Arduino. Pour le savoir il suffit d'écrire cette ligne de commande: `ls /dev/tty*`



# Bouton et LED

Les boutons sont aujourd'hui utilisés partout, que ce soit sur un système compliqué (voiture) ou simple (interrupteur). C'est pour cela qu'il est important de savoir gérer un bouton.

L'objectif sera de:

Commander une LED avec un bouton

Transmettre à l'utilisateur l'état de la LED

Faire clignoter une LED

La LED est une diode électroluminescente (DEL en français, ou LED en l'anglais : light-emitting diode). C'est un système qui émet de la lumière lorsqu'il est parcouru par un courant électrique. Une diode électroluminescente ne laisse passer le courant électrique que dans un seul sens. Si elle est branchée à l'envers alors elle ne fonctionnera jamais.



Figure 15 : symbole d'une LED

Sens de la LED: il y a trois manières de différencier l'anode et la cathode. Tout d'abord la partie tronquée est le moins. Ensuite la partie la plus grande à l'intérieur de la LED représente le moins. Enfin la plus grande patte représente le côté du plus.

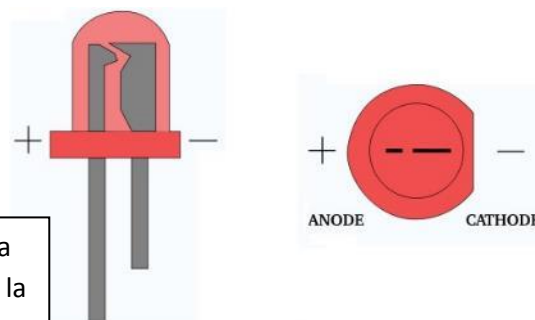


Figure 16 : différencier le + et le - d'une LED

Il faut s'imaginer pour cette mise en œuvre que la LED peut être remplacé par une lampe, un moteur... (Grâce à des relais ou non.) Il s'agit juste ici de faire fonctionner un système à l'aide d'un bouton. Nous en profiterons pour allumer-éteindre simplement la LED mais aussi le faire clignoter.

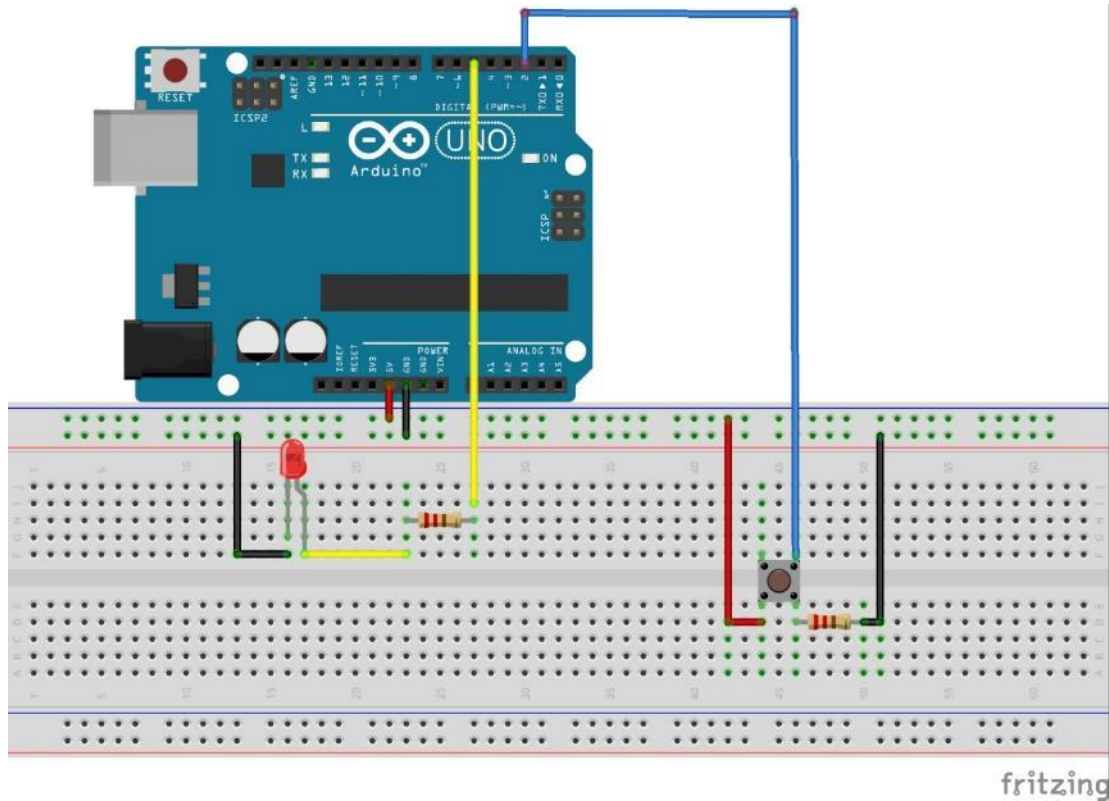
Tout d'abord le code pour allumer-éteindre une LED : voir Annexe

Ensuite le code pour faire clignoter une LED si l'on appuie sur le bouton : voir Annexe



Le montage ne changera pas pour les deux codes:

Il suffira d'un bouton poussoir, d'une carte Arduino, de deux résistances de 300  $\Omega$  et d'une LED rouge et d'une breadboard.



Pour juste allumer la LED lorsque l'on appuie sur le bouton :

```
int bouton = 2; // bouton sur la broche 2
boolean etatbouton = false; // on initialise la valeur de la variable pour
le bouton à 0
int led = 5; // la LED est branché sur la broche 5
void setup() {

  Serial.begin(9600); // moniteur série
  pinMode(bouton, INPUT); // on définit le bouton comme une entrée
  pinMode(led, OUTPUT); //on définit le LED comme une sortie
}

void loop() {

  etatbouton=digitalRead(bouton); // on lit la valeur du bouton est on la
stocke dans etatbouton
  Serial.println(etatbouton); // on écrit sur le moniteur l'état du bouton

  if (etatbouton == true){ // si le bouton est appuyé
    digitalWrite(led,HIGH); // on allume la LED
  }
  else{ // sinon
    digitalWrite(led,LOW); // on éteind la LED
  }
}
```



Pour faire clignoter la LED lorsque l'on appuie sur la LED :

```
int bouton = 2; // met le bouton sur la broche 2
boolean etatbouton = false; // initialisation de la variable du bouton à 0
int led = 5; // LED sur la broche 5
int compteur = 0; // initialisation du compteur à 0

void setup() {

  Serial.begin(9600); // on allume le moniteur série
  pinMode(bouton, INPUT); // bouton en entrée
  pinMode(led, OUTPUT); // led en sortie
}

void loop() {

  etatbouton=digitalRead(bouton); // on lit la valeur du bouton
  Serial.println(etatbouton); // on écrit la valeur du bouton sur le moniteur

  if (etatbouton == true){ // si le bouton est appuyé
    while(compteur != 10) //tant que compteur est différent de 10
    {
      compteur ++ ; //On incrémente d'un le compteur
      digitalWrite(led,HIGH); // on allume la LED
    }
    compteur = 0; // on remet le compteur à 0
  }
  else{ // sinon
    digitalWrite(led,LOW); // on éteint la lampe
  }
}
```



# La photo résistance

La photo résistance est une (ou cellule photoconductrice) est un composant électronique dont la résistivité varie en fonction de la quantité de la lumière incidente : plus elle est éclairée, plus sa résistivité baisse.

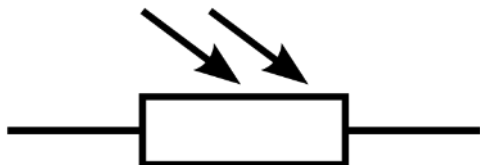


Figure 17 : symbole d'une photorésistance



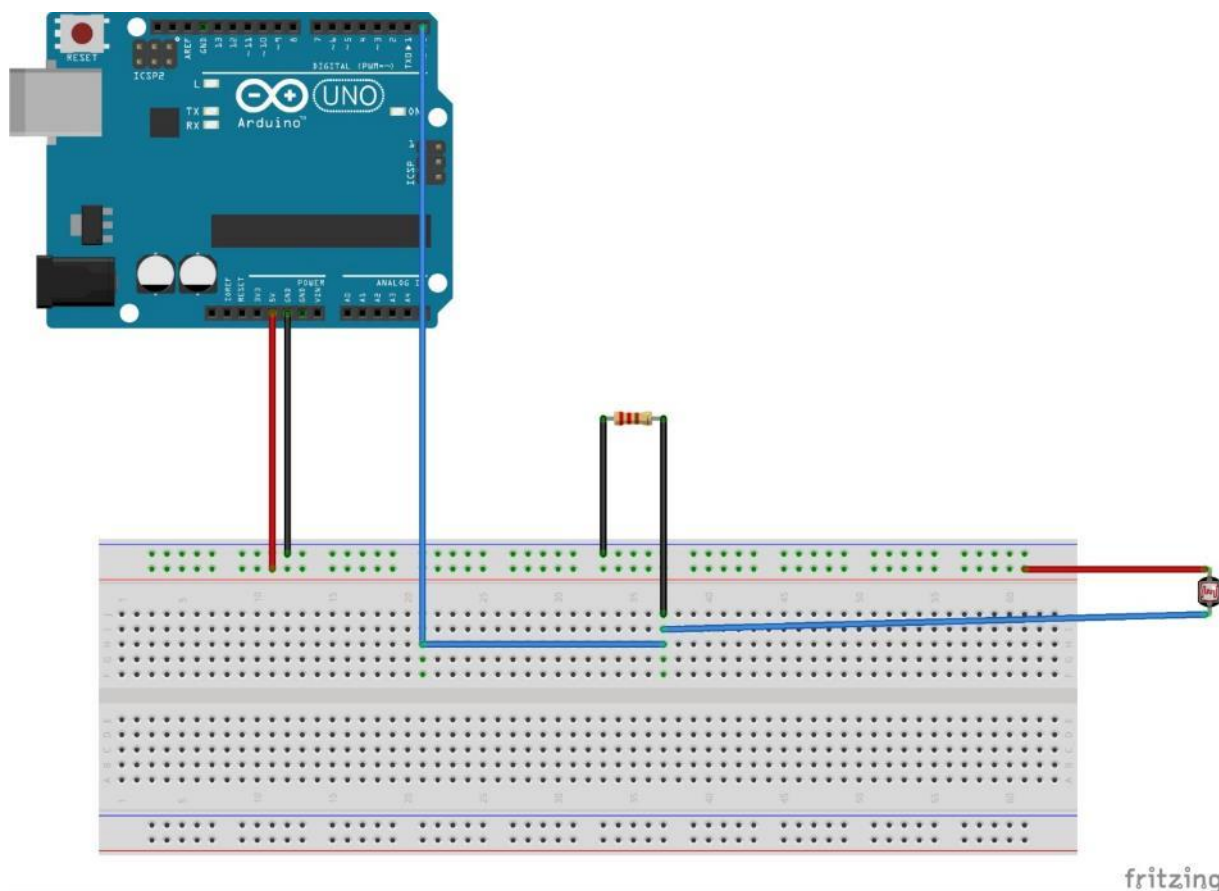
Figure 18 : photorésistance

À noter qu'il existe des petits modules déjà fait où l'on peut même régler la précision de la photorésistance.

L'objectif sera de :

Lire la valeur de la photorésistance

La transmettre à l'utilisateur



Pour ce montage il faut une Arduino, une photorésistance, une résistance, des fils, une breakboard.

Le code est le suivant :

```
int port = A0; // la broche A0 sera utilisé pour lire la valeur de la
photorésistance
int valeur = 0; // on initialise la valeur de la photorésistance à 0
int phot1 = 0; // variable initialisée à 0 qui sert à changer l'échelle de
valeur

void setup() {
    Serial.begin(9600); // on allume un moniteur série 9600 bauds
}

void loop() {
    valeur = analogRead(port); // on lit la valeur de la photorésistance que
l'on stocke dans valeur
    phot1 = map (valeur,0,1023,0,100); // on change l'échelle de valeur, 0 à
100 correspond à 0 à 1023 proportionnellement
    Serial.println('valeur phot1=',phot1); // on écrit sur le moniteur série
valeur phot1 = 5
    delay(1000); // on attend un seconde
}
// Remarque : 0 % correspond au noir et 100 % correspond à la lumière
```



# Soudage

J'ai dû souder pour assembler des fils pour permettre au thermocouple d'être correctement alimenté sur l'Arduino. En effet au bout du thermocouple il y avait des fils dénudés, mais les contacts ne sont pas corrects entre la breakboard Arduino et le fils Arduino, c'est pour cela que les fils Arduino sont à souder sur le thermocouple.

Pour réaliser ce montage, j'ai eu besoin, d'un fer à souder, d'un thermocouple, d'une pince à dénuder, de fils Arduino et d'étain, et de gaine thermo rétractable.

1. Dénuder les parties qui vont être soudé ensemble avec la pince à dénuder
2. Souder les fils ensemble avec le fer chaud et l'étain.
3. Mettre de la gaine thermo rétractable autour de chaque soudure pour les isoler les unes des autres.

Voilà le résultat :



Figure 20 : deux fils prêt à être soudés



Figure 19 : fils dénudé d'un côté

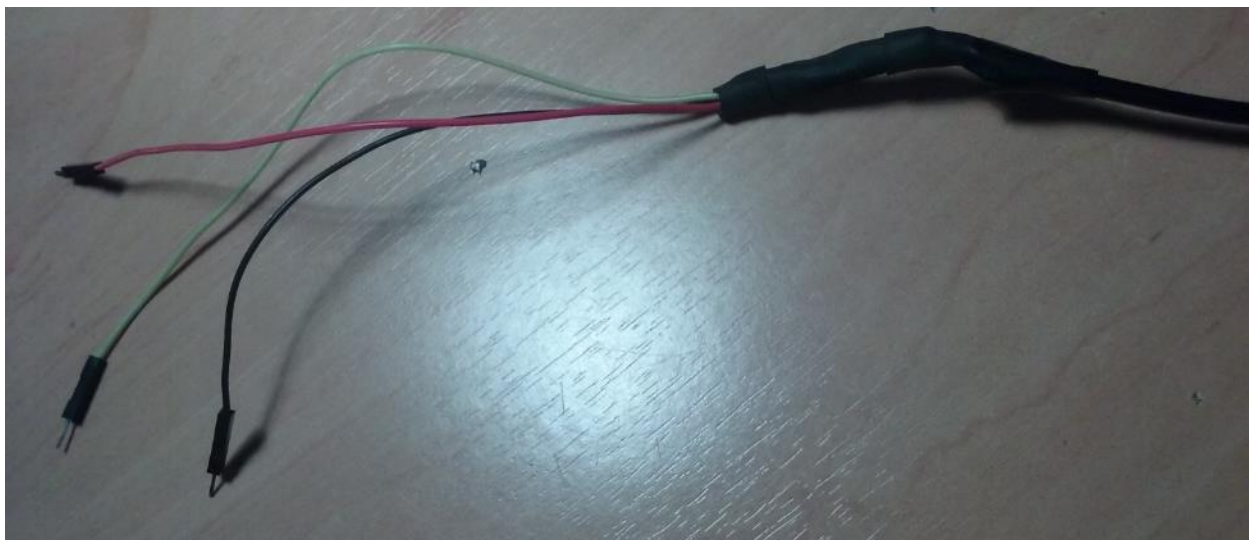
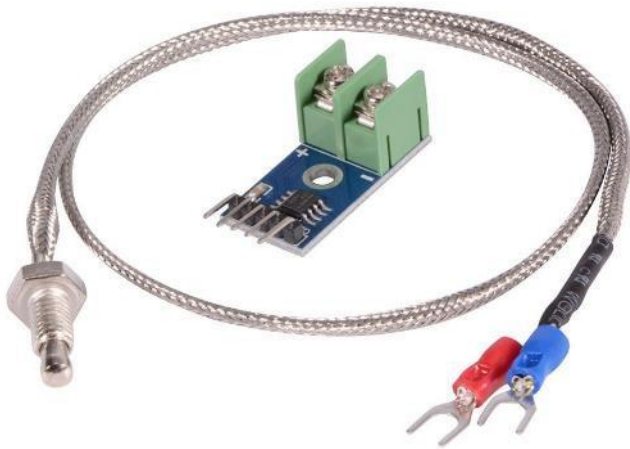


Figure 21 : résultat

# Le thermocouple

J'ai tout d'abord essayé d'alimenter le thermocouple avec l'Arduino, et de récupérer la valeur qui sortait du fil des données du thermocouple. Or je n'ai pas réussi à les lire. Le thermocouple délivre des tensions de l'ordre de 100 microvolts, il faut alors passer par un conditionneur afin de pouvoir lire une valeur.

J'ai donc commandé un thermocouple ainsi que son conditionneur MAX6675 (comme ici: [http://tiptopboards.free.fr/arduino\\_forum/viewtopic.php?f=2&t=75](http://tiptopboards.free.fr/arduino_forum/viewtopic.php?f=2&t=75) ) et j'essayerai de le mettre en œuvre aussi tôt que possible. Voici la photo du thermocouple et de son conditionneur :



Un thermocouple est un montage constitué de deux fils de métaux différents, soudés à l'une de leurs extrémités (jonction appelée soudure chaude). C'est partie est installée dans le milieu où il faut mesurer la température. Les deux autres extrémités sont à relier aux bornes d'un voltmètre, ici du conditionneur. C'est la soudure froide (ou soudure de référence).

Figure 22 : thermocouple et conditionneur



## Ecran LCD 16\*2

Il est très utile de pouvoir créer une interface de communication utilisateur/machine. En effet souvent il est important d'avoir un retour machine. Par exemple on aime connaître une valeur de mesure (température, vitesse...) on aime connaître si la machine fonctionne correctement et autre... C'est pour cela qu'un écran est essentiel ! Il existe énormément d'interface, les ordinateurs, les ordinateurs sur les machines, le moniteur série (Arduino)... Ici un écran LCD 16\*2 va me permettre d'afficher des petits messages, ou des températures, avec deux lignes de 16 caractères.



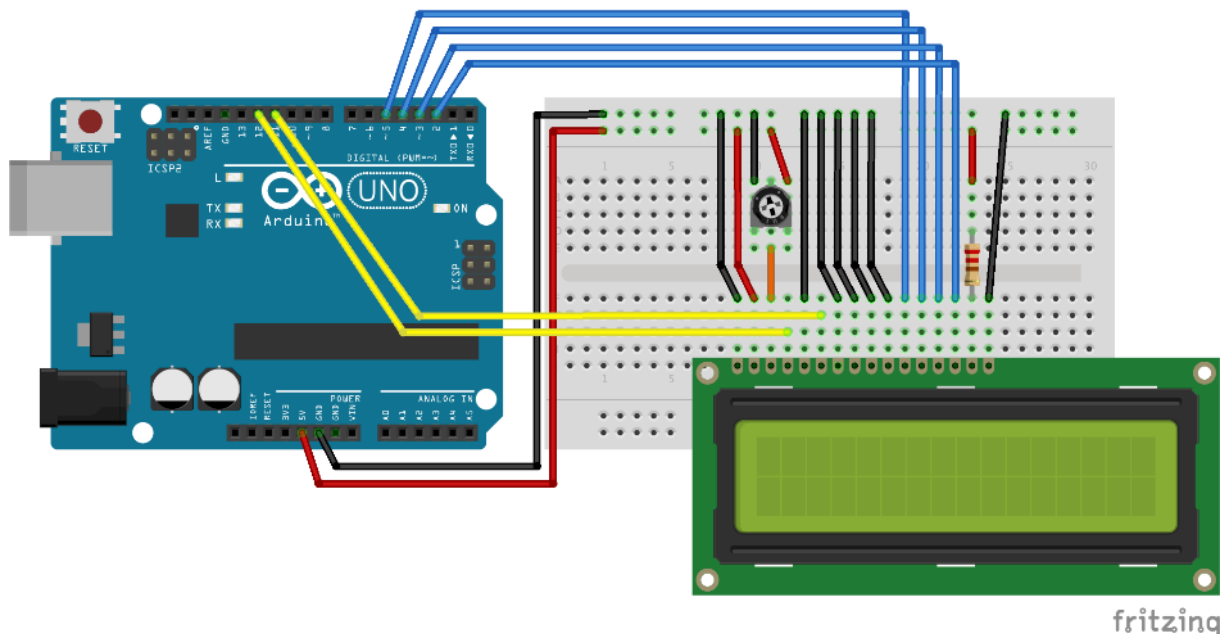
Figure 23 : écran LCD

L'objectif sera de :

Transmettre un message de l'Arduino vers l'utilisateur grâce à l'écran LCD  
Voir quelques fonctionnalités de l'écran

Le montage est le suivant:

Il est nécessaire d'avoir : un écran LCD 16\*2, une résistance de 220Ω, un potentiomètre, des fils, une Arduino est une résistance variable.



Le code est le suivant, les commentaires seront utiles pour comprendre ce que chaque commande effectue.

```
#include <LiquidCrystal.h> // Inclusion de la librairie pour afficheur LCD
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // définir les broches de l'Arduino
qui servent pour l'envoi des données vers l'écran LCD

void setup() {

    lcd.begin( 16, 2); // on définit un écran LCD 16*2
    lcd.clear(); // on nettoie l'écran

}

void loop() {

    // base
    lcd.print("MESSAGE"); // On écrit sur l'écran LCD message
    delay(1000); // on attends 1000 millisecondes (une seconde)
    lcd.clear(); // on efface l'écran du LCD

    lcd.write("personne detectee"); // on écrit sur l'écran LCD qu'une
    personne est detectée
    lcd.setCursor(0,1); // On passe une ligne sur l'écran LCD
    lcd.write("piece B"); // on écrit pièce B
    lcd.home(); // on remplace le curseur aux coordonnées (0,0)
    lcd.write (" Attention "); // on écrit attention sur l'afficheur
    lcd.noDisplay(); // on fait disparaître le texte
    delay(500); // on attend 500millisecondes
    lcd.display(); // on fait réapparaître le texte
    delay(500); // on attend 500 milliseconde
    lcd.clear(); // on nettoie l'écran
    lcd.noCursor(); // on cache le curseur
    lcd.cursor() ; // on ne cache plus le curseur
    lcd.blink() ; // on fait clignoter le curseur
    lcd.noBlink() ; // on arrête le clignotement du curseur
    lcd.scrollDisplayLeft() ; //décale l'affichage d'une colonne vers la
    gauche
    lcd.scrollDisplayRight() ; //décale l'affichage d'une colonne vers la
    droite

    // faire défiler le texte
    lcd.setCursor(16,0); // on place le curseur à la fin de la première ligne
    lcd.write("decalage"); // on décrit décalage en partant du curseur
    for(int x=0; x<16; x++) // pour décaler 16 fois
    {
        lcd.scrollDisplayLeft(); // on décale d'une colonne à chaque
        fois vers la gauche
        delay(250); // petite pause entre chaque décalage
    }

}
```

Remarque : c'est le seul programme que je n'ai pas pu tester car l'écran LCD ne marchait pas, j'en ai donc recommandé un, j'attends son arrivé...



# Node Red

Node-Red est un serveur Web consommant peu de ressources, capable de fonctionner sur un Raspberry. Il est très pratique car permet des mis en œuvre simple et rapide, et permet également une communication simplifié !

Hélas je n'ai pas réussi à l'utiliser pour l'instant, j'attends donc d'avoir une séance avec vous pour mieux comprendre.

## Objectifs futurs

J'ai pour but en poursuite de projet, de mettre en œuvre le thermocouple, comprendre Node-Red, améliorer l'aspect communication homme/machine. Stocker des mesures ou autre sur une carte SD, ou un dossier. Etre capable d'avoir un retour photo d'une pièce (surveillance...). De plus j'aimerais utiliser les relais pour commander des systèmes en 230V, et j'aimerais également commander des moteurs.



## Fiche Pratique

### Mettre du code sur Word avec la mise en page Arduino:

En effet je me suis aperçu que lorsque l'on copiait du code sur Word depuis Arduino, les couleurs Arduino n'était pas respectée. Ce qui est dérangement pour la compréhension du code. J'ai trouvé sur Internet un moyen simple et rapide de passer d'Arduino à Word. Tout d'abord copier le code Arduino en tant qu'HTML (onglet Edition puis copier en tant qu'HTML) puis le coller dans un bloc note. Ensuite ajouter au début ajouter <html><body> à la fin ajouter </body></html>. Sauvegarder en extension html. Enfin dans Word dans l'onglet insertion-objet-texte et sélectionner le fichier créer précédemment. Le texte apparaît alors avec sa mise en forme Arduino.

### Exporter une image Fritzing:

Il est possible pour insérer son image dans un Word de faire un "imprime écran" grâce à la touche "impécr" du montage Fritzing, et de le coller dans le document Word. Mais il est aussi possible depuis Fritzing de créer une image de son montage. Il suffit d'aller dans l'onglet Fichier, puis exporter, puis comme image et enfin choisir le format de l'image qui va être créée.

