Comment utiliser un écran OLED avec Raspberry Pi Pico

Table of Contents

Le Raspberry Pi Pico ne manque pas d'options en matière d'affichage numérique. On peut utiliser des écrans LCD, une sortie vers VGA / DVI ou utiliser des écrans sur mesure tels que l'écran Pico Display ou l'écran IPS de Pico Explorer Base. Mais parfois, on a besoin d'une petite option bon marché pour faire le travail. Les écrans OLED tels que le modèle de 0,96 pouce utilisé dans ce didacticiel sont simples à utiliser avec MicroPython et sont peut coûteux, ce qui les rend idéaux pour les projets.

Dans ce tutoriel, on va voir comment à connecter un écran OLED à un Raspberry Pi Pico via l'interface I2C, puis on va installer une librairie **MicroPython** via l'éditeur **Thonny** et l'utiliser pour écrire du texte à l'écran.

L'écran OLED utilise le protocole I2C pour s'interfacer avec le Raspberry Pi Pico. Ce qui signifie qu'on a seulement besoin.

- Un Raspberry Pi Pico exécutant MicroPython
- 4 fils de raccordement femelle à femelle
- Un écran I2C OLED

Ce mode d'emploi comprend des sections sur le dessin de formes et d'objets à l'écran, comment convertir des images bitmap pour les écrans OLED et comment animer des images.

Installation et configuration

Connecter un écran OLED au Raspberry Pi Pico

Le mappage des broches pour I2Cpour l'écran oled sh1107 est le suivant:

Id	broche	Pin Map I2C
3v	xxxxxx	Vcc
G	xxxxxx	Gnd
D2	GPIO 5	SCK / SCL
D1	GPIO 4	DIN / SDA
D0	GPIO 16	Res
G	xxxxxx	CS
G	xxxxxx	D/C

Il faut utiliser le câblage suivant:

1. Connecter le GND de l'écran à n'importe quel GND sur le Pico (fil noir).
2. Connecter VDD / VCC à 3V3 sur le Pico (fil rouge).
3. Connecter SCK / SCL à I2C0 SCL (GP1, broche physique 2, fil orange).
4. Connecter SDA à I2C0 SDA (GP0, broche physique 1, fil jaune).
5. Connecter votre Raspberry Pi Pico à votre ordinateur et ouvrez l'application Thonny.

Installation du driver

Avec le matériel connecté et Thonny ouvert, il faut maintenant installer une bibliothèque pour que Python puisse communiquer avec l'écran.

1. Cliquer sur Outils > Gérer les packages pour ouvrir le gestionnaire de packages de Thonny pour les bibliothèques Python.
2. Taper “sh1107” dans la barre de recherche et cliquer sur “Rechercher sur PyPI”.
3. Cliquer sur “micropython-sh1107” dans les résultats renvoyés, puis cliquer sur Installer. Cela copiera la bibliothèque dans un dossier, lib sur le Pico.
4. Cliquer sur Fermer pour revenir à l'interface principale.

Programmation d'un écran OLED sur Raspberry Pi Pico

Affichage d'un ligne de texte

Pour écrire une seule ligne de texte sur l'écran OLED, on n'a besoin que de six lignes de MicroPython.

Depuis la bibliothèque machine, importer les classes Pin et I2C. Ces derniers servent à communiquer avec l'écran OLED, rattaché au GPIO du Pico.

```
from machine import Pin, I2C
```

Importer la bibliothèque d'écrans OLED.

```
from ssd1306 import SSD1306_I2C
```

Créer un objet, i2c, qui stocke le canal I2C utilisé, dans ce cas zéro, les broches SDA et SCL auxquelles l'écran est connecté, et enfin la fréquence à laquelle on se connecte à l'écran OLED.

```
i2c=I2C(0,sda=Pin(0), scl=Pin(1), freq=400000)
```

Créer un objet, oled, qui sera utilisé pour communiquer avec l'écran OLED. Il a trois arguments, la largeur et la hauteur de l'écran (128 x 64 pixels) et les détails de connexion I2C.

```
oled = SSD1306_I2C(128, 64, i2c)
```

Écrire une ligne de texte en haut à gauche de l'écran, position 0,0.

```
oled.text("Ceci est un test", 0, 0)
```

Enfin, utiliser la commande show pour rendre la sortie à l'écran.

```
oled.show()
```

Le code final devrait ressembler à ceci

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C

i2c=I2C(0,sda=Pin(0), scl=Pin(1), freq=400000)
oled = SSD1306_I2C(128, 64, i2c)

oled.text("Tom's Hardware", 0, 0)
oled.show()
```

Dans Thonny, cliquer sur `Fichier >> Enregistrer` et enregistrer le fichier sur le Raspberry Pi Pico sous **oled-test.py**, puis cliquer sur le bouton de lecture vert pour lancer le code et le texte apparaîtra sur l'écran OLED.

Dessiner des formes simples sur des écrans OLED avec Pico

Des formes et des lignes simples peuvent être dessinées à l'écran avec une seule commande. Chacune de ces commandes aura besoin de **oled.show()** pour être vue.

La plupart de ces méthodes ont un paramètre de couleur mais, avec un écran monochrome, on utilisera toujours une couleur de « 1 » (0 signifie pixel désactivé).

- **oled.pixel(x,y,c)**: dessine un pixel à la position x,y et utilise c pour définir la couleur du pixel,

1 étant allumé, 0 étant éteint. Exemple: `oled.pixel(10,10,1)`
- **oled.hline(x,y,w,c)**: trace une ligne horizontale à partir du point x,y qui a une largeur définie (w) en pixels et une couleur ©. Exemple: `oled.hline(2,3,4,1)`
- **oled.vline(x,y,h,c)**: trace une ligne verticale à partir du point x,y qui a une hauteur définie (h) en pixels et une couleur ©. Exemple: `oled.vline(0,0,64,1)`
- **oled.line(x1,y1,x2,y2,1)** : trace une ligne diagonale des points x1, y1 à x2, y2 avec la couleur ©. Exemple: `oled.line(0,0,128,64,1)`
- **oled.rect(x,y,w,h,c)** : dessine un rectangle commençant au point x.y et pour une largeur (w) et une hauteur (h) définies. Utilise © pour définir la couleur des pixels. Par exemple: `oled.rect(0,0,64,32,1)`
- **oled.fill_rect(x,y,w,h,c)**: dessine un rectangle rempli à partir du point x.y et pour une largeur (w) et une hauteur (h) définies, utilise © pour définir la couleur des pixels. Par exemple: `oled.fill_rect(0, 0, 64, 32, 1)`

Affichage des graphiques sur l'écran OLED avec Pico

On peut afficher bien plus que du texte à l'écran. En utilisant une technique intelligente, on peut convertir une image JPEG en une chaîne d'octets.

On va utiliser le code ci-dessus comme base de travail. Donc, si on ne l'a pas déjà fait, copie et coller le code final ci-dessus dans **Thonny**.

Importer la bibliothèque **framebuf**. Cette bibliothèque permet au code de créer des images bitmap et de les afficher à l'écran.

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import framebuf
```

Créer un nouvel objet, TH qui stockera un tableau d'octets qui composent l'image. Pour l'instant laisser le contenu du tableau vide, on le remplira de blanc plus tard.

```
i2c=I2C(0,sda=Pin(0), scl=Pin(1), freq=400000)
oled = SSD1306_I2C(128, 64, i2c)
TH = bytearray()
```

Créer un objet, FB, qui chargera l'image dans le framebuffer. On passe le nom de l'objet **bytearray**, les dimensions de l'image (64 x 64 pixels) puis on configure l'image pour qu'elle soit une image monochrome 1 bit.

```
fb = framebuf.FrameBuffer(TH,64,64, framebuf.MONO_HLSB)
```

Effacer l'écran, puis blit l'image sur l'écran. Utiliser ensuite show pour mettre à jour l'écran. Le blitting dessine l'image sur l'écran, dans ce cas il place l'image 64 x 64 au centre de l'écran. Où 32 est la position horizontale (x) et 0 est la position verticale (y).

```
```

```
oled.fill(0)
oled.blit(fb,32,0)
oled.show()
```

L'objet, TH, n'a actuellement aucune image à afficher. Pour créer un bytearray d'une image, il faut une image appropriée. L'écran mesure 128 x 64 pixels, mais une image de 64 x 64 pixels s'intégrera parfaitement au centre de l'écran. L'image doit être au format JPEG.

Pour convertir l'image en bytearray, on utilise le script python img2bytearray de Don Hui (Novaspirit).

1. Télécharger et extraire l'archive ZIP sur la machine. Cela crée un dossier, `img2bytearray` qui stocke le fichier Python.
2. Copier l' image dans le dossier img2bytearray.
3. Ouvrir une fenêtre Invite de commandes / Terminal et accéder au dossier `img2bytearray`.
4. Pour convertir l'image, on va appeller la commande img2bytearray. On doit fournir trois paramètres supplémentaires. Le premier est le nom du fichier image, les deux suivants sont les dimensions de l'image, dans ce cas 64 par 64 pixels: `python3 img2bytearray.py VOTRE-IMAGE.jpg 64 64`
5. La commande génère un flux d'octets. Copier le texte de b'...' et le coller à l'intérieur de la parenthèse de l'objet TH: `TH = bytearray(b’...’)`

Le code final devrait ressembler à ceci

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import framebuf

i2c=I2C(0,sda=Pin(0), scl=Pin(1), freq=400000)
oled = SSD1306_I2C(128, 64, i2c)
TH = bytearray(b’...’)

fb = framebuf.FrameBuffer(TH,64,64, framebuf.MONO_HLSB)
oled.fill(0)
oled.blit(fb,32,0)
oled.show()
```

Dans Thonny, cliquer sur `Fichier >> Enregistrer` et enregistrer le fichier sur le Raspberry Pi Pico sous **oled-test.py**, puis cliquer sur le bouton de lecture vert pour lancer le code et l'image apparaîtra sur l'écran OLED.

Animation de graphiques sur l'écran OLED avec Pico

Créer une animation avec l'écran OLED consiste à déplacer ou à modifier des objets pour donner l'illusion de mouvement. Pour ce faire, il faut utilisé des tableaux de deux octets, pour le marteau (TH) et le logo (LOGO).

Encore une fois, on se basera sur le code des exemples précédents.

1. Importer les bibliothèques de code précédentes.

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import framebuf
```

1. Configurer les broches I2C pour l'écran OLED.

```
i2c=I2C(0,sda=Pin(0), scl=Pin(1), freq=400000)
oled = SSD1306_I2C(128, 64, i2c)
```

1. Créer une boucle while True pour exécuter le code en continu.

```
while True:
```

1. Créer le tableau d'octets TH. pour y stocker les octets qui composent une icône.

```
  TH = bytearray(b'...')
```

1. Créer un objet, FB, qui chargera l'image dans le framebuffer, puis effacera l'écran. Passer le nom de l'objet bytearray, les dimensions de l'image (64 x 64 pixels) puis configurer l'image pour qu'elle soit une image monochrome 1 bit.

```
  fb = framebuf.FrameBuffer(TH,64,64, framebuf.MONO_HLSB)
  oled.fill(0)
```

1. Utiliser une boucle for avec une plage de -64 à 128, pour créer une animation de glissement de base pour le marteau. La valeur négative masque le marteau à gauche de l'écran, faisant défiler lentement le marteau de gauche à droite jusqu'à ce qu'il sorte de la plage visible. En changeant la valeur de “i” dans **oled.blit()** on créer l'illusion de mouvement.

```
  for i in range(-64,128):
      oled.blit(fb,i,0)
      oled.show()
```

1. Créer un nouveau tableau d'octets pour le logo.

```
   LOGO = bytearray(b'...')
```

1. Créer un objet, FB, qui chargera l'image dans le framebuffer, puis effacera l'écran. Passer le nom de l'objet bytearray, les dimensions de l'image (128 x 64 pixels) puis configurer l'image pour qu'elle soit une image monochrome 1 bit.

```
fb = framebuf.FrameBuffer(LOGO,128,64, framebuf.MONO_HLSB)
```

1. Le code restant est en grande partie le même qu'avant, la seule différence étant que la plage de la boucle for passe de -128 à 128 pour s'adapter au logo défilant sur l'écran.

```
   oled.fill(0)
```

```
    for i in range(-128,128):
        oled.blit(fb,i,0)
        oled.show()
```

Le code devrait ressembler à ceci

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import framebuf


i2c=I2C(0,sda=Pin(0), scl=Pin(1), freq=400000)
oled = SSD1306_I2C(128, 64, i2c)

while True:
   TH = bytearray(b'...')
   fb = framebuf.FrameBuffer(TH,64,64, framebuf.MONO_HLSB)
   oled.fill(0)
   for i in range(-64,128):
       oled.blit(fb,i,0)
       oled.show()
   LOGO = bytearray(b'...')
   fb = framebuf.FrameBuffer(LOGO,128,64, framebuf.MONO_HLSB)
   oled.fill(0)
   for i in range(-128,128):
       oled.blit(fb,i,0)
       oled.show()
```

Dans **Thonny**, cliquer sur `Fichier >> Enregistrer` et enregistrer le fichier sur votre Raspberry Pi Pico sous **oled-test.py**, puis cliquer sur le bouton de lecture vert pour lancer le code et l'animation défilera sur l'écran OLED.

Menu avec interraction des boutons

Créer un menu dans lequel les options peuvent être interagies avec des boutons et les afficher sur l'écran LCD

Le programme ci-dessous crée une première couche du menu.

```
from RPLCD import CharLCD, cleared, cursor
import RPi.GPIO as GPIO
import time
from gpiozero import Button
GPIO.setmode(GPIO.BCM)
GPIO.setup(5, GPIO.OUT)
GPIO.setup(22, GPIO.OUT)
GPIO.setup(17, GPIO.OUT)
GPIO.setup(27, GPIO.OUT)
```

```
GPIO.setup(12, GPIO.OUT)
GPIO.setup(25, GPIO.OUT)
GPIO.setup(24, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)
GPIO.setup(26, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(19, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(20, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(21, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

lcd = CharLCD(numbering_mode=GPIO.BCM, cols=16, rows=2, pin_rs=13, pin_e=6,
pins_data=[5,22,17,27,12,25,24,23])

Nbutton = Button(19, pull_up=False, bounce_time=0.001)
Ubutton = Button(26, pull_up=False, bounce_time=0.001)

def Fruits():
    lcd.cursor_pos = (0, 0)
    lcd.write_string("Fruits-1")
    lcd.cursor_pos = (1, 0)
    lcd.write_string("Fruits-2")
def Vegetables():
    lcd.cursor_pos = (0, 0)
    lcd.write_string("Vegetables-1")
    lcd.cursor_pos = (1, 0)
    lcd.write_string("Vegetables-2")
def Clothes():
    lcd.cursor_pos = (0, 0)
    lcd.write_string("Clothes-1")
    lcd.cursor_pos = (1, 0)
    lcd.write_string("Clothes-2")
def Shoes():
    lcd.cursor_pos = (0, 0)
    lcd.write_string("Shoes-1")
    lcd.cursor_pos = (1, 0)
    lcd.write_string("Shoes-2")

count=0
try:
    while 1:
        for count in range(0,4):
            Nbutton.wait_for_press()
            count = count +1
            print (count, "presses so far")
            if count == 1:
                Fruits()
            elif count == 2:
                Vegetables()
            elif count == 3:
                Clothes()
            elif count == 4:
                Shoes()
```

```
            else:
                pass
            time.sleep(0.3)
except:
    pass

finally:
    lcd.clear()
    GPIO.cleanup()
```

Navigation dans menu

Le programme ci-dessous utilise 4 boutons représentés dans le code par la saisie du clavier:

1. le bouton 1 déplace “>” vers le haut
2. le bouton 2 déplace “>” vers le bas
3. le bouton 3 sélectionne l'option où le pointeur “>” est
4. le bouton 4 revient au menu principal

```
def level0():
    #main menu
    lcd.cursor_pos = (0, 2)
    lcd.write_string('main menu option 1')
    lcd.cursor_pos = (1, 2)
    lcd.write_string('main menu option 2')
    lcd.cursor_pos = (2, 2)
    lcd.write_string('main menu option 3')
    lcd.cursor_pos = (3, 2)
    lcd.write_string('main menu option 4')
def level1():
    lcd.cursor_pos = (0, 2)
    lcd.write_string('sub menu 1 option 1')
    lcd.cursor_pos = (1, 2)
    lcd.write_string('sub menu 1 option 2')
    lcd.cursor_pos = (2, 2)
    lcd.write_string('sub menu 1 option 3')
    lcd.cursor_pos = (3, 2)
    lcd.write_string('sub menu 1 option 4')
def level2():
    lcd.cursor_pos = (0, 2)
    lcd.write_string('sub menu 2 option 1')
    lcd.cursor_pos = (1, 2)
    lcd.write_string('sub menu 2 option 2')
    lcd.cursor_pos = (2, 2)
    lcd.write_string('sub menu 2 option 3')
    lcd.cursor_pos = (3, 2)
    lcd.write_string('sub menu 2 option 4')
def level3():
    lcd.cursor_pos = (0, 2)
```

```
    lcd.write_string('sub menu 3 option 1')
    lcd.cursor_pos = (1, 2)
    lcd.write_string('sub menu 3 option 2')
    lcd.cursor_pos = (2, 2)
    lcd.write_string('sub menu 3 option 3')
    lcd.cursor_pos = (3, 2)
    lcd.write_string('sub menu 3 option 4')
def level4():
    lcd.cursor_pos = (0, 2)
    lcd.write_string('sub menu 4 option 1')
    lcd.cursor_pos = (1, 2)
    lcd.write_string('sub menu 4 option 2')
    lcd.cursor_pos = (2, 2)
    lcd.write_string('sub menu 4 option 3')
    lcd.cursor_pos = (3, 2)
    lcd.write_string('sub menu 4 option 4')
def clearpointer():
    lcd.cursor_pos = (0, 0)
    lcd.write_string(' ')
    lcd.cursor_pos = (1, 0)
    lcd.write_string(' ')
    lcd.cursor_pos = (2, 0)
    lcd.write_string(' ')
    lcd.cursor_pos = (3, 0)
    lcd.write_string(' ')
level = 0
position = 0


level0()
lcd.cursor_pos = (position, 0)
lcd.write_string('>')
while True:

    button = input('1 = down, 2 = up, 3 = select, 4 = level up ')
    if button = 1:
        position = position + 1

    if button = 2:
    position = position - 1
    clearpointer()
    lcd.cursor_pos = (position, 0)
    lcd.write_string('>')
    if button = 3:
        if position = 0:
            level = 1
            lcd.clear()
            level1()
            position = 0
        if position = 1:
            level = 2
```

```
            lcd.clear()
            level2()
            position = 0
        if position = 2:
            level = 3
            lcd.clear()
            level3()
            position = 0

        if position = 3:
            level = 4
            lcd.clear()
            level4()
            position = 0
    if button = 4:
            level = 0
            lcd.clear()
            level0()
            position = 0
```

On commence donc au menu principal et on charge l'écran LCD avec ses options telles que définies dans la fonction level0 puis on place le pointeur sur la première ligne en plaçant le curseur à 0,0 et en écrivant la chaîne “>”.

maintenant on peut déplacer le “>” vers le bas en saisissant 1 et l'instruction if correspondante ajoute 1 à la variable de position et efface le “>” en appelant la fonction clearpointer puis redessine le “>” à la nouvelle position.

la même chose se produit si on déplace le “>” vers le haut en entrant 2 mais on soustrait 1 de la variable de position.

Le programme ne sait pas ce qu'il y a à l'écran uniquement à quel niveau il se trouve et la position du pointeur “>”, en utilisant les variables niveau et position. Il n'y a pas de vérification d'erreur dans cet exemple pour empêcher le “>” de sortir de l'écran et de planter le programme.

maintenant, si on entre un 3, le programme prend la position actuelle et à partir de là décide avec quel sous-menu mettre à jour l'écran, en effaçant d'abord l'écran puis en appelant la fonction appropriée, les options du niveau 1 au niveau 4,il remet ensuite le “>” à la position 0 et écrit la chaîne.

enfin, dans cet exemple, la saisie de 4 entraîne le retour du menu au menu principal et le “>” à la position 0.

Utilisation de la Bibliothèque RPLCD

En temps normal, la bibliothèque censée être utilisée pour envoyer des informations aux broches

GPIO est celle livrée de base avec Raspbian : RPi.GPIO. Son utilisation est très proche du système et nécessite de manipuler des octets pour indiquer à l'écran LCD les caractères à envoyer. Fort heureusement, une autre bibliothèque existe déjà et simplifie grandement toute cette partie. Il s'agit de RPLCD.

L'utilisation de la bibliothèque s'articule autour de la classe CharLCD. Cette classe se compose de trois fonctions essentielles :

- **clear()** nettoie l'écran des caractères affichés.
- **write_string()** envoie un caractère à l'écran.
- **close()** ferme le canal de communication avec l'écran. Cette fonction accepte un booléen, clear, qui lorsqu'il est affecté de la valeur True, nettoie l'écran avant de fermer la connexion.

Positionner le texte

Plutôt que de simplement afficher du texte sur l'écran en partant de la première case (ligne 0, colonne 0), on peut demander de commencer ou on veut. On utilise pour ceci l'instruction lcd.cursor_pos = (ligne, colonne). Attention, en informatique, on commence toujours à compter par 0 et non par 1. Si on veut écrire sur la première ligne, ligne=0 alors que si on veut écrire sur la deuxième ligne : ligne=1 et c'est la même chose pour les colonnes.

Prenons donc l'exemple où on veut écrire Raspberry sur la deuxième ligne en laissant deux cases vides avant le mot. On aura donc `LCD-positionTexte.py`:

```
from RPLCD.gpio import CharLCD

lcd = CharLCD(cols=16, rows=2, pin_rs=37, pin_e=35, pins_data=[33, 31, 29,
23])

lcd.cursor_pos = (1, 2)
lcd.write_string(u'Raspberry')
view raw
```

Les paramètres donnés à la fonction **CharLCD()** renseignent la librairie sur le type d'écran utilisé (16×02), et les numéros de pins sur lesquels l'écran est relié.

Remettre l'écran à zéro (effacer les caractères)

Si on souhaite changer un mot à l'écran ou simplement ne plus rien afficher, on va utiliser la fonction **lcd.clear()**. Celle-ci ne prend pas d'arguments et efface tous les caractères qui sont affichés à l'écran. (L'écran reste tout de même allumé). On va pouvoir par la suite utiliser cette fonction pour faire clignoter du texte à l'écran. `LCD-remettreEcranAZero.py` permet d'afficher un texte pendant 5 secondes. Il faut par conséquent importer le module time.

```
from RPLCD.gpio import CharLCD
import time
```

```
lcd = CharLCD(cols=16, rows=2, pin_rs=37, pin_e=35, pins_data=[33, 31, 29,
23])

lcd.write_string(u'Espace RaspberryFrancais')
time.sleep(5)
lcd.clear()
view raw
```

Faire clignoter du texte

LCD-clignoterTexte.py utilise la fonction vue précédemment pour facilement faire clignoter du texte à l'écran, avec une boucle while:

```
from RPLCD.gpio import CharLCD
import time

lcd = CharLCD(cols=16, rows=2, pin_rs=37, pin_e=35, pins_data=[33, 31, 29,
23])

while True :
    lcd.write_string(u'Espace Raspberry')
    time.sleep(2)
    lcd.clear()
    time.sleep(1)
view raw
```

Passer une ligne

Lorsqu'un texte est trop long, la librairie continue le texte à la ligne suivante. Si l'on souhaite forcer le passage de ligne, il suffit de rajouter \n\r à l'endroit voulu. Voici un exemple `LCD-passerLigne.py`:

```
from RPLCD.gpio import CharLCD

lcd = CharLCD(cols=16, rows=2, pin_rs=37, pin_e=35, pins_data=[33, 31, 29,
23])

lcd.write_string(u'Bonjour\n\ra tous !!!')
view raw
```

Afficher la date et l'heure

La première chose qu'il peut être intéressant de faire est d'afficher la date actuelle sur l'écran. Pour ceci, `LCD-date&Heure.py` utilise simplement la bibliothèque time de Python.

```
from RPLCD.gpio import CharLCD
import time

lcd = CharLCD(cols=16, rows=2, pin_rs=37, pin_e=35, pins_data=[33, 31, 29,
23])

while True:
    lcd.cursor_pos = (0, 0)
    lcd.write_string("Date: %s" %time.strftime("%d/%m/%Y"))
    lcd.cursor_pos = (1, 0)
    lcd.write_string("Heure: %s" %time.strftime("%H:%M:%S"))
view raw
```

Afficher l'adresse IP

Pour afficher l'adresse IP du Raspberry Pi à l'écran, `LCD-adresseIP.py` récupére l'adresse IP à l'aide d'une fonction, et affiche la valeur de sortie sur l'écran.

En fonction de si on est connecté en Wifi ou par câble, il faut mettre l'argument de **obtenirIP()** respectivement sur **wlan0** ou **eth0**.

```
from RPLCD.gpio import CharLCD
import socket
import fcntl
import struct

lcd = CharLCD(cols=16, rows=2, pin_rs=37, pin_e=35, pins_data=[33, 31, 29,
23])

def obtenirIP(nom):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    return socket.inet_ntoa(fcntl.ioctl(
        s.fileno(),
        0x8915,
        struct.pack('256s', nom[:15])
    )[20:24])

lcd.write_string(u"Adresse IP :")

lcd.cursor_pos = (1, 0)
lcd.write_string(obtenirIP('wlan0')) #Si connection Ethernet :
obtenirIP('eth0')
view raw
```

Afficher des caractères personnalisés

En plus des caractères de l'alphabet, on peut afficher ce qu'on veut sur chacune des 32 cases (16×2). Chacune de ces case est en réalité une matrice 5×8. Voici le code qui permet d'afficher une case remplie en damier et une autre qui affiche un smiley.

```
from RPLCD.gpio import CharLCD
from RPLCD import cleared, cursor

lcd = CharLCD(cols=16, rows=2, pin_rs=37, pin_e=35, pins_data=[33, 31, 29,
23])

unSurDeux = (
    0b10101,
    0b01010,
    0b10101,
    0b01010,
    0b10101,
    0b01010,
    0b10101,
    0b01010,
)
smiley = (
    0b00000,
    0b00000,
    0b01010,
    0b01010,
    0b00000,
    0b10001,
    0b01110,
    0b00000,
)

lcd.create_char(0, unSurDeux)
lcd.create_char(1, smiley)

lcd.write_string(unichr(0))
lcd.write_string(unichr(1))
view raw
```

Menu déroulant

On n'utilise pas gpiozero donc les boutons sont juste définis dans RPi.GPIO en utilisant l'option de détection d'événement, et on n'a pas besoin de configurer les broches gpio pour l'écran LCD car les sorties RPLCD s'en charge.

On utilise donc 3 boutons:

1. le bouton 1 sélectionne la 1ère option sur chaque menu (gpio 26)
2. le bouton 2 sélectionne la 2ème option sur chaque menu (gpio 19)
3. le bouton 3 sélectionne les étapes de retour dans le menu un niveau à la fois (gpio 20)

```
from RPLCD.gpio import CharLCD
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

GPIO.setup(26, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)# button 1
GPIO.setup(19, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)# button 2
GPIO.setup(20, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)# button 3
GPIO.setup(21, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) # not used

lcd = CharLCD(numbering_mode=GPIO.BCM, cols=16, rows=2, pin_rw=None,
pin_rs=13, pin_e=6, pins_data=[5,22,17,27,12,25,24,23],
charmap='A02', auto_linebreaks=False)

# set global variables used
update = 1 # causes LCD to be updated while set to 1
mlevel = 1 # current menu level
blevel = 1 # last menu level

def level1():
    #main menu
    lcd.cursor_pos = (0, 2)
    lcd.write_string("1.Footware")
    lcd.cursor_pos = (1, 2)
    lcd.write_string("2.Transport")
def level2():
    #sub menu
    lcd.cursor_pos = (0, 2)
    lcd.write_string("1.Shoes")
    lcd.cursor_pos = (1, 2)
    lcd.write_string("2.Boots")
def level3():
    #sub menu
    lcd.cursor_pos = (0, 2)
    lcd.write_string("1.Ground")
    lcd.cursor_pos = (1, 2)
    lcd.write_string("2.Air")
def level4():
    #sub menu
    lcd.cursor_pos = (0, 2)
    lcd.write_string("1.Red")
    lcd.cursor_pos = (1, 2)
    lcd.write_string("2.Black")
def level5():
```

```
    #sub menu
    lcd.cursor_pos = (0, 2)
    lcd.write_string("1.Lace up")
    lcd.cursor_pos = (1, 2)
    lcd.write_string("2.Zip up")
def level6():
    #sub menu
    lcd.cursor_pos = (0, 2)
    lcd.write_string("1.Car")
    lcd.cursor_pos = (1, 2)
    lcd.write_string("2.Bus")
def level7():
    #sub menu
    lcd.cursor_pos = (0, 2)
    lcd.write_string("1.Plane")
    lcd.cursor_pos = (1, 2)
    lcd.write_string("2.Helicopter")
def option1(channel):
    global mlevel, update, blevel
    blevel = mlevel
    mlevel = mlevel*2
    update = 1
def option2(channel):
    global mlevel, update, blevel
    blevel = mlevel
    mlevel = (mlevel*2) + 1
    update = 1
def goback(channel):
    global mlevel, update, blevel
    mlevel = blevel
    blevel = int(mlevel/2)
    update = 1
GPIO.add_event_detect(26, GPIO.RISING, callback=option1, bouncetime=200)
GPIO.add_event_detect(19, GPIO.RISING, callback=option2, bouncetime=200)
GPIO.add_event_detect(20, GPIO.RISING, callback=goback, bouncetime=200)

#loop to update menu on LCD
while True:
    while update ==0:
        time.sleep (0.1)
    lcd.clear()
    if mlevel == 1: level1()
    if mlevel == 2: level2()
    if mlevel == 3: level3()
    if mlevel == 4: level4()
    if mlevel == 5: level5()
    if mlevel == 6: level6()
    if mlevel == 7: level7()
    update = 0
```

Il n'y a pas de vérification d'erreur dans le programme, donc si on essaye de sélectionner une option au-delà du 3e niveau, le programme échouera, de même si on essaye de revenir au-delà du menu principal.

La mise à jour de l'affichage est gérée par la boucle while true , qui a une attente créée par la seconde boucle while qui ne permet de mettre à jour l'affichage que si quelque chose change, comme lorsqu'on appuie sur un bouton.

Maintenant, chaque bouton appelle sa propre fonction:

1. pour les boutons 1 et 2, on fait monter à travers les niveaux en utilisant quelques calculs simples pour que les niveaux s'incrémentent correctement, mais parce qu'on met à jour les variables **level** , **blevel** et **update** dans les fonctions, il faut utiliser les variables globales utilisées dans la boucle principale qui met à jour l'affichage.
2. le bouton 3 fonctionne légèrement différemment car il réinitialise d'abord la variable **mlevel** au niveau b, puis calcule ce que devrait être le nouveau niveau b, de sorte qu'on puisse continuer à appuyer sur le bouton pour descendre dans les niveaux du menu.