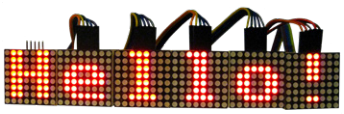
MicroPython pour l'ESP32 et ses copains

Partie 2 : piloter facilement les écrans matriciels

Günter Spanner (Allemagne)

Le premier article de cette série portait sur l'interface de développement et les commandes simples. Passons maintenant aux applications pratiques. Par exemple, avec des bibliothèques de programmes appropriées, fabriquer et programmer des écrans matriciels à LED de grande taille devient un jeu d'enfant. Outre le défilement de textes, vous pouvez également afficher l'heure ou la température locale.



Le mini-écran du premier article [1] était plutôt destiné aux plus petits appareils. Dans cet article, nous montrons comment réaliser un afficheur de grande surface avec l'aide de Python. Les dispositifs choisis ici sont des matrices de LED de 8×8 = 64 LED individuelles chacune. En joignant plusieurs unités, il est possible de construire des afficheurs de taille presque illimitée. Ils peuvent servir comme téléscripteurs pour les cours de bourse ou panneaux d'affichage de résultats sportifs à la maison. Avec quelques instructions en Python, on peut créer de grands afficheurs pour l'heure ou la température, qui peuvent parfaitement servir à des fins publicitaires à de multiples occasions.

Afficheurs à matrice de points

Les afficheurs matriciels permettent de produire n'importe quels symboles et graphiques simples en plus de l'affichage de chiffres et de lettres. Ils sont donc plus intéressants que les afficheurs classiques à sept segments, qui ne permettent que l'affichage des chiffres de 0 à 9. En outre, une fois spécifiée, la taille des afficheurs à sept segments est fixe. Les matrices de points, en revanche, peuvent être mises à l'échelle comme on le souhaite.

Il est aussi possible d'obtenir une luminosité bien supérieure à celle des écrans OLED. De plus, des surfaces d'affichage de plusieurs mètres de long et de large peuvent être réalisées avec des matrices de LED. On utilise souvent cette catégorie d'afficheurs pour des panneaux d'affichage ou de publicité de grande surface dans des endroits très fréquentés, gares ou aéroports, bus et trains, etc. Dans les salles des marchés boursiers du monde entier, ils affichent les cours des actions sous forme de téléscripteurs. Par ailleurs, les matrices de points sont particulièrement répandues en Asie,

Figure 1. Les matrices de points à LED sont très populaires en Asie.

car elles peuvent afficher sans problème les caractères extrême-orientaux, comme le montre la **figure 1**.

Pilotage

Les matrices de LED avec 5×7 = 35 LED sont très utilisées. Cependant, une grande variété d'autres modèles sont également disponibles. On trouve beaucoup d'éléments avec 8×8 LED. La commande directe de 8×8 points nécessiterait 65 lignes, car il faudrait raccorder 64 anodes et une cathode commune. Avec une matrice, on peut s'en sortir avec beaucoup moins de connexions. Ici, 8 LED sont connectées ensemble en lignes et en colonnes. Ainsi, seules 8 + 8 = 16 connexions sont nécessaires. La **figure 2** montre le principe pour une matrice de 3×4. Au lieu de 13 connexions pour la commande individuelle des LED, 7 connexions suffisent ici. Par exemple, pour allumer la 2e LED de la 2e ligne de la matrice, toutes les lignes, sauf la seconde, doivent être soumises au potentiel HIGH. La deuxième ligne, en revanche, doit être au potentiel GND. Pour les colonnes, seule la seconde doit être au potentiel HIGH.

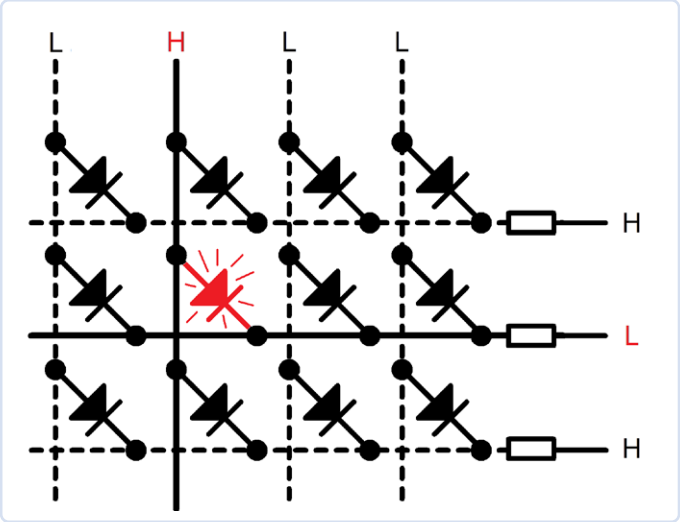


Figure 2. Principe de commande des matrices de points.

Le pilotage direct des matrices de points mobilise une grande partie des ressources du contrôleur. S'il faut également enregistrer les valeurs de capteurs ou commander des actionneurs, même un contrôleur ESP32 puissant atteint rapidement ses limites. La commande d'écrans plus grands, comportant une centaine de diodes électroluminescentes ou plus, deviendrait également rapidement un problème – d'une part en raison de la puissance de calcul requise, d'autre part en raison du nombre limité de broches disponibles sur un contrôleur. En outre, la vitesse d'exécution relativement faible du code Python aurait également ici un effet négatif. Il est donc conseillé d'utiliser des puces pilotes d'écran bon marché, telles que le MAX7219. Ces composants ont une interface compatible SPI et peuvent piloter avec seulement trois broches des écrans qui comportent jusqu'à 8×8 = 64 éléments de matrice. L'interface périphérique série (SPI) est très répandue et fréquemment utilisée, notamment dans l'électronique grand public. Le système de bus et surtout son utilisation sous MicroPython sont décrits en détail dans un livre Elektor de l'auteur [2].

On trouve des modules qui regroupent le pilote et la matrice. La **figure 3** en montre un exemple.

La connexion des modules pilotes à l'ESP32 est très simple. Seules les trois broches SPI doivent être reliées à la carte à contrôleur :

- MAX7219_data (DIN) Port D02
- MAX7219_load (CS) Port D05
- MAX7219_clock (CLK) Port D04
- MAX7219_GND ESP32 GND

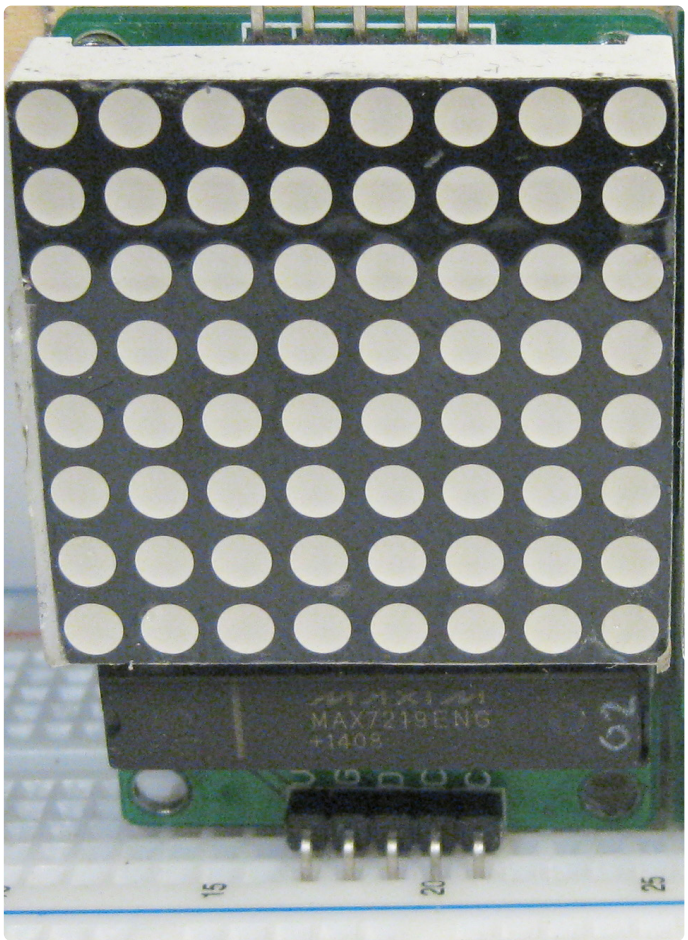


Figure 3. Module matriciel à LED.

À cause de la consommation électrique relativement élevée, il est recommandé d'avoir une alimentation externe. Notez que la tension d'alimentation des modules doit être d'environ 3,3 à 4 V pour garantir la compatibilité avec l'ESP32. Ainsi, d'une part, les modules reçoivent toujours la tension d'alimentation minimale admissible selon la fiche technique. D'autre part on évite une surtension sur les entrées du contrôleur. Il est également possible d'utiliser un convertisseur de niveau bidirectionnel 3,3/5 V.

Il faut aussi prévoir des condensateurs de blocage suffisants. Des lignes VCC et de masse à faible résistance sont également essentielles. Si vous prenez ces précautions, vous pouvez construire des écrans de presque toutes les tailles. Le contrôleur n'est

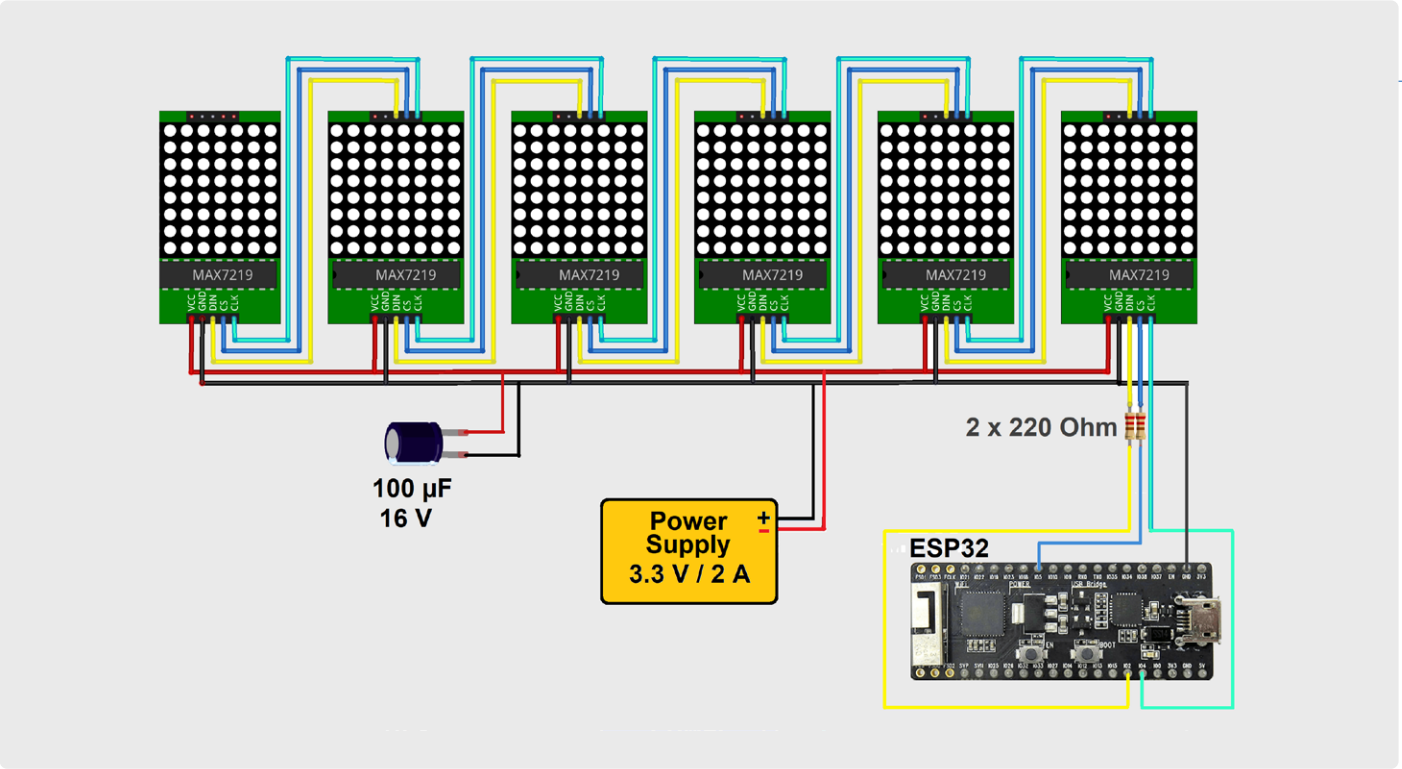


Figure 4. Regroupement de modules matriciels à LED.

pratiquement plus chargé, car il ne fait qu'envoyer des commandes simples via le bus SPI. En outre, suffisamment de broches restent libres pour gérer des capteurs externes ou d'autres périphériques. La réalisation d'écrans de grande surface, par ex. à des fins publicitaires ou pour des événements sportifs, n'est donc plus un obstacle. La **figure 4** montre le schéma du circuit complet pour réaliser un afficheur matriciel à six chiffres avec un module à contrôleur ESP32.

Réussir avec la bonne bibliothèque

Vous trouverez dans le paquet à télécharger en [4] un lien pour une bibliothèque Python spécialisée dans la commande des circuits intégrés Maxim. Après avoir chargé le fichier pilote *Max7219.py* sur le contrôleur, les instructions suivantes sont disponibles :

```
spi = SPI(1, baudrate=10000000, polarity=1, phase=0,
sck=Pin(CLK), mosi=Pin(DIN))
ss = Pin(CS, Pin.OUT)
```

Pour cela il faut affecter les broches ainsi :

CLK = 4, DIN = 2 et CS = 5

Ensuite, un objet d'affichage peut être créé :

```
display = max7219.Matrix8x8(spi, ss, MN)
```

où MN est le nombre d'éléments de matrice utilisés. Ensuite, les textes et les graphiques peuvent être dessinés à l'aide des commandes du **listage 1.** Il est ainsi possible de créer des publicités efficaces, faciles à lire même à plusieurs mètres de distance.

Matrice de LED en action

Le code suivant montre un exemple d'application pour un affichage avec six éléments de matrice 8 x 8 :

```
# LED_matrix_test.py

import max7219
from machine import Pin, SPI

spi = SPI(1, baudrate=10000000, polarity=1, phase=0,
sck=pin(4), mosi=pin(2))
ss = Pin(5, Pin.OUT)
```

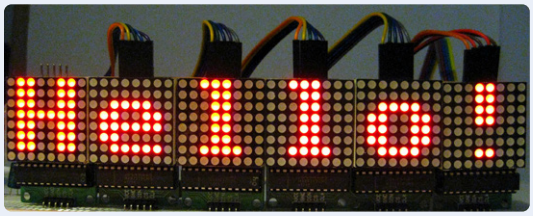


Figure 5. Affichage matriciel par points avec six éléments de matrice 8×8.

Figure 6. Affichage matriciel par points avec douze éléments de matrice.

Listage 1. Commandes d'affichage.

```
display.pixel(x,y,1)        # définit un pixel aux coordonnées x, y
display.pixel(x,y,0)        # supprime un pixel aux coordonnées x, y
display.hline(x,y,l,1)      # ligne horizontale de x, y - longueur l
display.vline(x,y,l,1)      # ligne verticale de x, y - longueur l
display.line(a,b,c,d,1)     # ligne de a, b à c, d
display.rect(a,b,c,d,1)     # rectangle avec coins a, b, c, d
display.text('Text',x,y,1)  # texte à la position x, y
display.scroll(x,0)         # défilement par x pixels
display.show()              # rafraîchit l'affichage
```

```
display = max7219.Matrix8x8(spi, ss, 6)
display.text('Python',0,0,1)
display.show()
```

Après avoir chargé le programme dans le contrôleur ESP, le texte s'affiche sur l'écran (**fig. 5**).
L'afficheur peut être facilement agrandi avec des modules de matrice supplémentaires. La **figure 6** montre une matrice de douze éléments. L'affichage n'est pas limité aux symboles et textes statiques puisque Python permet de mettre en oeuvre sans problème des graphiques en mouvement. La section suivante en présente un exemple.

Téléscripteurs boursiers et autres

La fonction de défilement peut servir à créer des téléscripteurs. Il est ainsi possible d'afficher des textes plus longs même sur des écrans petits ou courts. Le programme du **listage 2** fait défiler le texte « Python » de droite à gauche sur un écran matriciel.
Les lettres individuelles sont ainsi transmises avec

```
display.text('...',40,0,1)
```

et créées sur le bord droit de l'écran. Elles sont ensuite déplacées vers la gauche de la valeur `pixelDistance` avec la fonction `moveLeft` :

```
for i in range(8):
    display.scroll(-1,0)
    sleep(speedFactor
```

La valeur `speedFactor` permet de modifier la vitesse d'exécution. Admirez le résultat sur YouTube [3].

Affichage de la température en grand format

De nombreuses pharmacies, banques et autres boutiques tentent d'attirer l'attention de clients potentiels en faisant de la publicité avec de grands afficheurs de température ou d'heure. Ces afficheurs attirent également l'attention lors de manifestations sportives, de foires commerciales, d'expositions et dans les FabLabs. Pour réaliser un afficheur de température, par exemple, il suffit d'ajouter le capteur correspondant à la configuration présentée ci-dessus. Un modèle particulièrement adapté est le DS18x20 de Maxim Integrated (anciennement Dallas). Avec la bibliothèque Python appropriée, le capteur peut être lu sans problème.
Ces capteurs communiquent via le bus 1-Wire et n'occupent donc qu'une seule broche d'E/S de l'ESP32. En outre, une bibliothèque Python prête à l'emploi est disponible pour cette série de capteurs. En plus du capteur lui-même, il faut juste une résistance de rappel de 4,7 kΩ. En utilisant le rappel interne de l'ESP32, il est même possible de l'omettre. Le capteur présente les caractéristiques suivantes :

Listage 2. Téléscripteur.

```
LED_matrix_ticker.py

import max7219
from machine import Pin, SPI
from time import sleep

spi = SPI(1,baudrate=10000000, polarity=1, phase=0,
sck=Pin(4), mosi=Pin(2))
ss = Pin(5,Pin.OUT)
speedFactor=0.05
pixelDistance=7
display = max7219.Matrix8x8(spi, ss, 6)

def moveLeft(Pixel):
   for i in range(Pixel):
      display.scroll(-1,0)
      sleep(speedFactor)
      display.show()

while True:
   display.text('P',40,0,1)
   moveLeft(pixelDistance)
   display.text('y',40,0,1)
   moveLeft(pixelDistance)
   display.text('t',40,0,1)
   moveLeft(pixelDistance)
   display.text('h',40,0,1)
   moveLeft(pixelDistance)
   display.text('o',40,0,1)
   moveLeft(pixelDistance)
   display.text('n',40,0,1)
   moveLeft(pixelDistance)
   display.text(' ',40,0,1)
   moveLeft(pixelDistance)
```

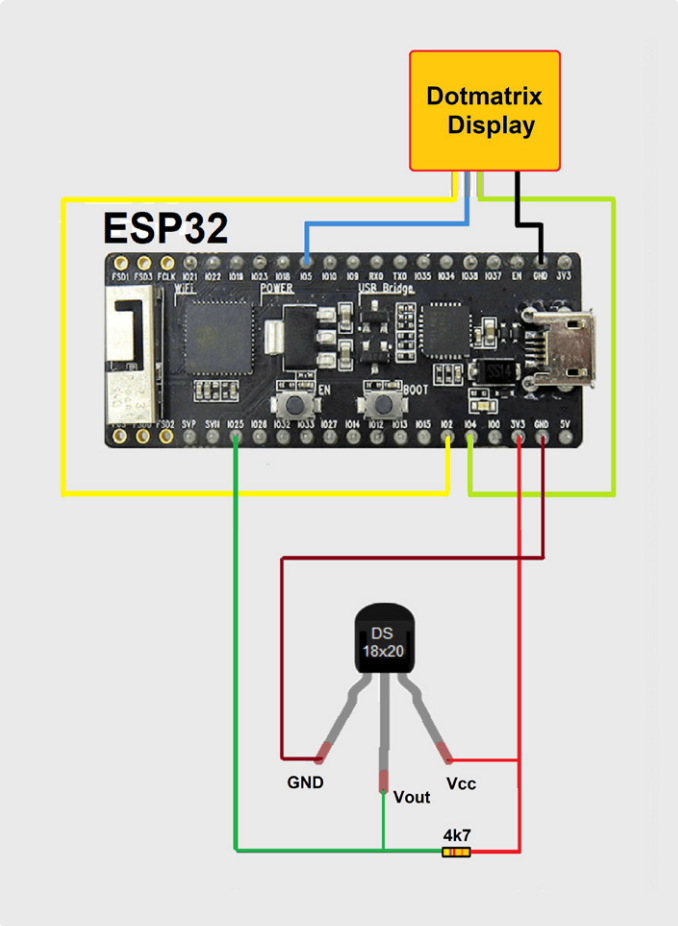


Figure 7. Capteur thermique DS18x20 relié au contrôleur ESP32.

- Tension d'alimentation : 3,0 V à 5,5 V
- Plage de température : −55 °C à +125 °C
- Précision de mesure : ±0,5 °C (−10 °C à +85 °C)
- Résolution : 9 bits, correspondant à environ 1/10 °C
- Durée de la période de mesure : 750 ms (max.)

Une broche connectée au bus 1-Wire permet de récupérer les mesures de plusieurs capteurs. En effet le protocole spécial du bus 1-Wire permet d'interroger en parallèle presque n'importe quel nombre de capteurs de température via une seule broche du contrôleur. Cependant, nous nous contentons ici d'un seul capteur. La **figure 7** montre la connexion du DS18x20 à l'ESP32. Un programme d'évaluation pour le DS18x20 affiche les valeurs de température enregistrées sur la console :

```
DS18x20_TempSens_demo.py

from machine import pin
import onewire
import ds18x20
import time

ow = onewire.OneWire(Pin(25)) # initialise le bus one
wire
ow.scan()
ds=ds18x20.DS18X20(ow) # crée l'objet ds18x20

while True:
    units=ds.scan() # recherche les unités ds18x20
    ds.convert_temp() # convertit la température
    for unit in units:
        print(ds.read_temp(unit)) # affichage
        time.sleep(1)
        print()
```

Le micrologiciel MicroPython contient par défaut les modules de lecture du capteur.
L'extension du programme à l'écran matriciel LED se fait en quelques lignes, voir le **listage 3**. La **figure 8** montre le résultat. On peut trouver dans le livre [2] d'autres détails sur la connexion de divers capteurs d'intensité lumineuse, d'humidité ou de champs magnétiques, entre autres.

Graphiques animés

Outre les lettres et les données numériques, des graphiques et même des animations peuvent être présentés sur la matrice de LED. Le programme du **listage 4** fait apparaître n'importe quel graphique en pixels sur l'écran.
Le graphique est défini dans l'image matricielle `icon`. Les points à allumer sont marqués d'un « 1 », les points sombres d'un « 0 » :

```
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 1, 0, 0, 1, 1, 0],
[0, 1, 1, 0, 0, 1, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 1, 0],
[0, 0, 1, 1, 1, 1, 0, 0],
```

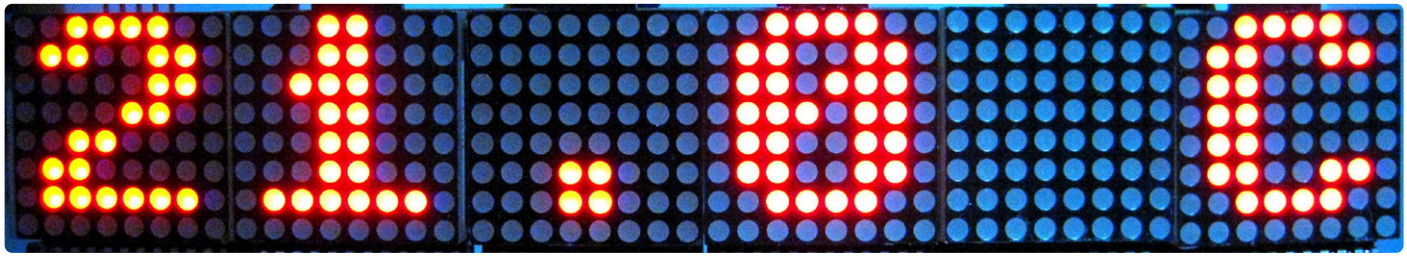


Figure 8. Un méga-afficheur de température.

Listage 3. Affichage de la température.

```
# DS18x20_to_LED_matrix.py

import max7219
from machine import Pin, SPI
import onewire
import ds18x20
import time

# pin assignment for LED matrix
spi = SPI(1, baudrate=10000000, polarity=1, phase=0, sck=Pin(4), mosi=Pin(2))
ss = Pin(5, Pin.OUT)

ow = onewire.OneWire(Pin(25))                    # initialise le bus 1-Wire
ow.scan()
ds=ds18x20.DS18X20(ow)                           # recherche les unités ds18x20

while True:
   units=ds.scan()                               # recherche les unités ds18x20
   ds.convert_temp()                             # convertit la température
   for unit in units:
      T=ds.read_temp(unit)
      output="T = {:5.1f} °C"                    # crée la chaîne formatée
      print(output.format(T))
      output=str(T)
      output="{:4.1f} C"                         # crée la chaîne formatée pour la matrice de LED
      display = max7219.Matrix8x8(spi, ss, 6)
      display.text(output.format(T),0,0,1)       # écrit la température sur la matrice de LED
      display.show()
   time.sleep(1)
```

De cette façon, n'importe quelle icône peut être créée. La fonction `enumerate` convertit le bitmap en un graphique affichable :

```
for y, row in enumerate(icon):
    for x, c in enumerate(row):
        display.pixel(x, y, c)
```

On obtient ainsi la **figure 9** sur la matrice de LED. On peut alors utiliser la fonction de défilement de la boucle principale pour déplacer sur l'afficheur l'image créée ainsi (voir également la vidéo YouTube [3]).

Résumé et perspectives

Après avoir présenté les commandes élémentaires dans le premier article de la série [1], cette deuxième partie permet de les mettre en œuvre dans diverses applications pratiques. De puissantes bibliothèques permettent de réaliser des projets impressionnants avec seulement quelques lignes de code. Cela confirme la potentiel remarquable de MicroPython également pour les applications de contrôleurs. Vous trouverez de plus amples informations et de nombreux projets pratiques dans le livre *MicroPython for Microcontrollers* [2]. La commande de servomoteurs, la transmission de données sans fil par RFID, le protocole MQTT et la transmission de mesures de capteurs sur l'internet y sont également traités en détail.

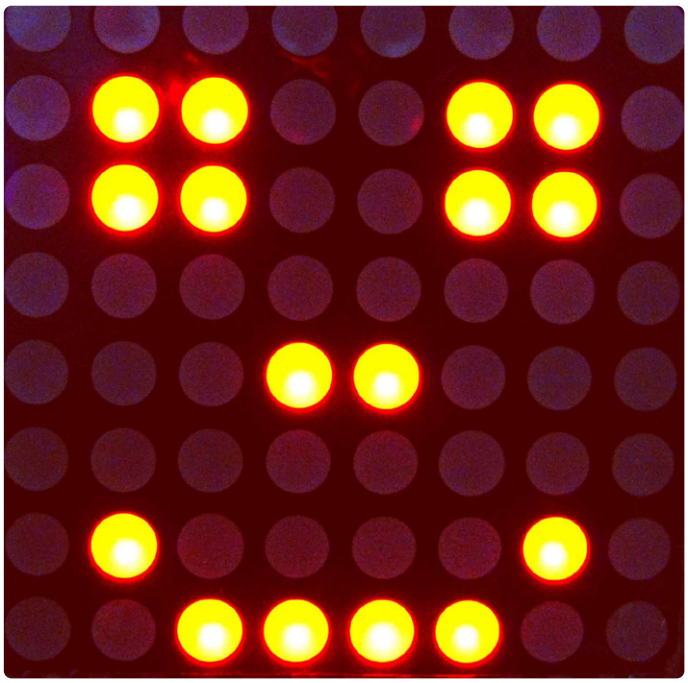
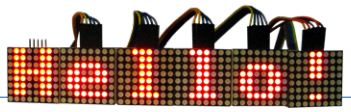


Figure 9. Graphique avec des pixels, fait main, sur l'écran matriciel.



Listage 4. Graphique en pixels.

```
#LED_matrix_icon.py

import max7219
from machine import Pin, SPI
from time import sleep

# sck -> serial clock - CLK; mosi -> master out slave in - DIN
# ss -> chip select - CS
spi=SPI(1,baudrate=10000000,polarity=1,phase=0,sck=Pin(4),mosi=P
in(2))
ss=Pin(5,Pin.OUT)

display=max7219.Matrix8x8(spi,ss,6)
display.brightness(0)

icon = [
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 1, 0, 0, 1, 1, 0],
     [0, 1, 1, 0, 0, 1, 1, 0],
     [0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 1, 1, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0],
     [0, 1, 0, 0, 0, 0, 1, 0],
     [0, 0, 1, 1, 1, 1, 0, 0],
]

for y, row in enumerate(icon):
   for x, c in enumerate(row):
     display.pixel(x, y, c)

display.show()

while True:
   for i in range(40):       # déplace le carré de gauche à droite
      display.show()
      display.scroll(1, 0)
      sleep(0.05)

   for i in range(40):       # déplace le carré de gauche à droite
      display.show()
      display.scroll(-1, 0)
      sleep(0.05)
```

Contributeurs

Texte et illustrations : **Günter Spanner**
Rédaction : **Jens Nickel,C.J. Abate**
Mise en page : **Harmen Heida**
Traduction : **Denis Lafourcade**

En outre, la combinaison de Python avec le domaine en pleine expansion de l'apprentissage automatique et de l'intelligence artificielle permettra, dans un avenir proche, des applications qui étaient auparavant impensables. Les nouvelles générations de contrôleurs, comme le Kendryte K210, permettent déjà l'évaluation directe des signaux audio et vidéo. Avec les algorithmes de traitement d'images basés sur Python, un large éventail d'options tournées vers l'avenir s'ouvre aux développeurs.

(210179-B-04)

Des questions, des commentaires ?
Contactez Elektor (redaction@elektor.fr).

LIENS

[1] **Günter Spanner, « MicroPython pour l'ESP32 et ses copains », Elektor 07-08/2021** : www.elektormagazine.fr/200179-04
[2] **Livre en anglais de G. Spanner, *MicroPython for Microcontrollers*, Elektor 2020** : www.elektor.fr/micropython-for-microcontrollers
[3] **Vidéo de démonstration sur YouTube** : http://www.youtube.com/watch?v=5uligjyKMEk
[4] **Téléchargement du logiciel** : http://www.elektormagazine.fr/210179-B-04