Bienvenue !

Nous vous remercions d'avoir acheté notre écran AZ-Delivery 1,3” OLED I2C. Dans les pages suivantes, vous apprendrez à utiliser et à configurer cet appareil pratique.

Amusez-vous bien !

Sommaire

Introduction

OLED signifie "Organic Light Emitting Diodes" (diodes électroluminescentes organiques). Les écrans OLED sont des réseaux de LEDs empilées ensemble dans une matrice. L'écran OLED 1.3 possède 128x64 pixels (LEDs). Pour contrôler ces LED, nous avons besoin d'un circuit de pilotage ou d'une puce. L'écran possède une puce pilote appelée SH1106. La puce pilote possède une interface I2C pour communiquer avec le microcontrôleur principal.

L'écran OLED et la puce pilote SH1106 fonctionnent dans la gamme 3.3V. Mais il y a un régulateur de tension 3.3V intégré, donc ces écrans peuvent être utilisés dans la gamme 5V.

Les performances de ces écrans sont bien meilleures que celles des écrans LCD traditionnels. La communication I2C simple et la faible consommation d'énergie les rendent plus adaptés à une variété d'applications.

Caractéristiques

Tension d'alimentation	de 3,3 V à 5 V
Interface de communication	I2C
Couleur du pixel	Blanc
Température de fonctionnement	de -20 à 70 ℃
Faible consommation d'énergie	< 11mA
Dimensions	36 x 34 x 3mm [1.4 x1.3 x 0.1”]

Pour prolonger la durée de vie de l'écran, il est courant d'utiliser un "économiseur d'écran". Il est recommandé de ne pas utiliser d'informations constantes sur une longue période, car cela réduirait la durée de vie de l'écran et augmenterait l'effet dit de "brûlure d'écran".

Comment configurer l'IDE Arduino

Si l'IDE Arduino n'est pas installé, suivez le *link* et téléchargez le fichier d'installation pour le système d'exploitation de votre choix

Pour les utilisateurs de Windows, double-cliquez sur le fichier .exe téléchargé et suivez les instructions de la fenêtre d'installation.

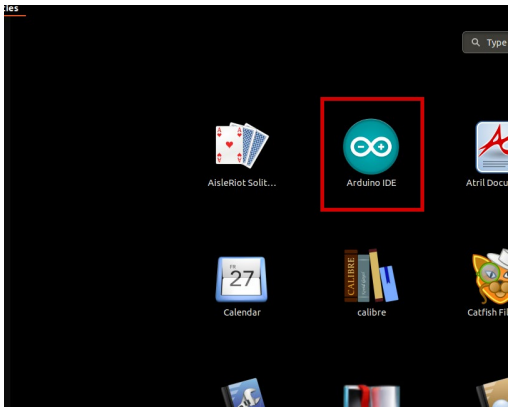
Pour les utilisateurs de Linux, téléchargez un fichier portant l'extension *.tar.xz*, qui doit être extrait. Lorsqu'il est extrait, allez dans le répertoire extrait et ouvrez le terminal dans ce répertoire. Deux scripts .sh doivent être exécutés, le premier appelé arduino-linux-setup.sh et le second appelé *install.sh*.

Pour exécuter le premier script dans le terminal, ouvrez le terminal dans le répertoire extrait et exécutez la commande suivante :
`sh arduino-linux-setup.sh user_name`

`user_name` - est le nom d'un superutilisateur dans le système d'exploitation Linux. Un mot de passe pour le superutilisateur doit être saisi au moment du lancement de la commande. Attendez quelques minutes pour que le script complète tout.

Le deuxième script appelé script *install.sh* doit être utilisé après l'installation du premier script. Exécutez la commande suivante dans le terminal (répertoire extrait) : **`sh install.sh`**

`Après l'installation de ces scripts, allez dans le dossier All Apps, où est installé l'IDE Arduino.`

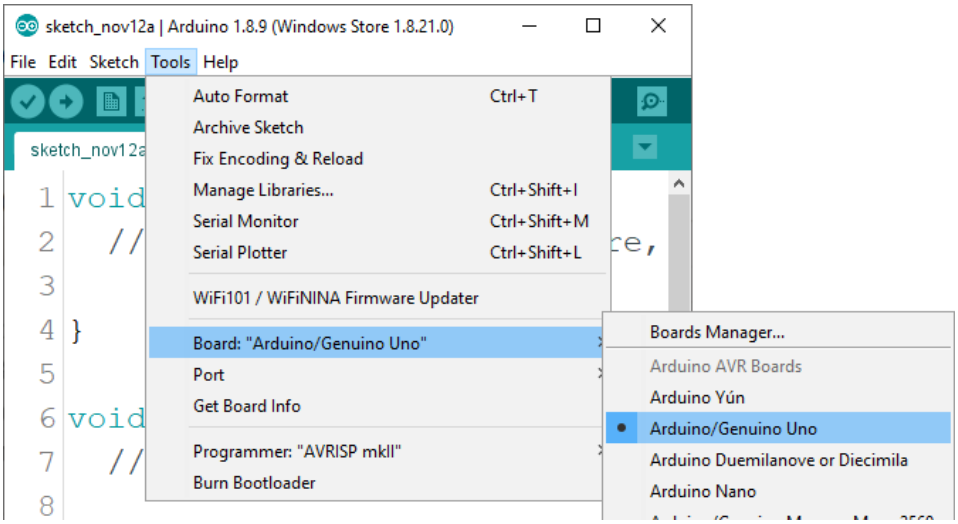


Presque tous les systèmes d'exploitation sont livrés avec un éditeur de texte préinstallé (par exemple, Windows est livré avec Notepad, Linux Ubuntu avec Gedit, Linux Raspbian avec Leafpad, etc.) Tous ces éditeurs de texte conviennent parfaitement à l'objectif de l'eBook.

La prochaine étape est de vérifier si votre PC peut détecter une carte Atmega328P. Ouvrez l'IDE Arduino fraîchement installé, et allez dans :

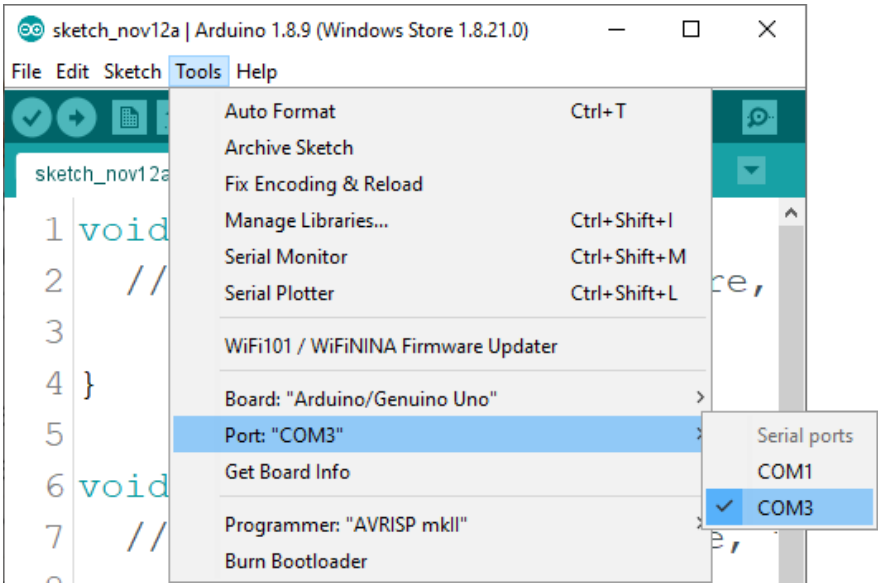
Tools > Board > {votre nom de conseil ici}

{votre nom de conseil ici} devrait être l'Arduino/Genuino Uno, comme on peut le voir sur l'image suivante :



Le port auquel la carte Atmega328P est connectée doit être sélectionné. Allez dans : Outils > Port > {le nom de la porte va ici}
et lorsque la carte Atmega328P est connectée au port USB, le nom du port peut être vu dans le menu déroulant de l'image précédente.

Si l'Arduino IDE est utilisé sous Windows, les noms des ports sont les suivants :



Pour les utilisateurs de Linux, par exemple, le nom du port est /dev/ttyUSBx, où x représente un nombre entier entre 0 et 9.

Comment configurer le Raspberry Pi et Python ?

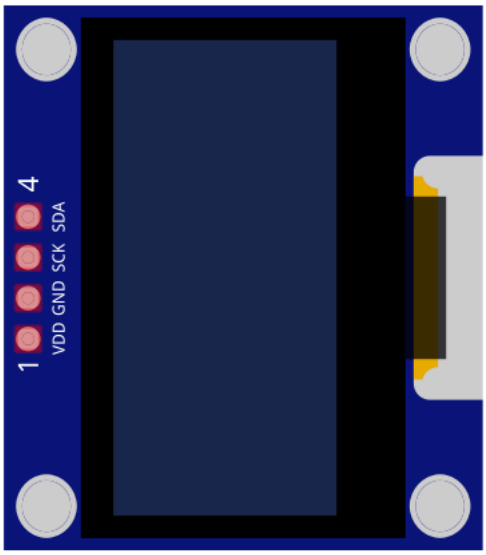
Pour le Raspberry Pi, il faut d'abord installer le système d'exploitation, puis tout configurer pour qu'il puisse être utilisé en mode Headless. Le mode Headless permet de se connecter à distance au Raspberry Pi, sans avoir besoin d'un écran de PC, d'une souris ou d'un clavier. Les seuls éléments utilisés dans ce mode sont le Raspberry Pi lui-même, l'alimentation électrique et la connexion Internet. Tout ceci est expliqué en détail dans le livre électronique gratuit :
Raspberry Pi Quick Startup Guide

Le système d'exploitation Raspbian est livré avec Python préinstallé.

Le pinout

L'écran OLED de 1,3” possède quatre broches. Le brochage est montré sur l'image suivante :

I2C Serial Data Line - SDA
I2C Serial Clock Line - SCK
Ground - GND
Power Supply - VDD

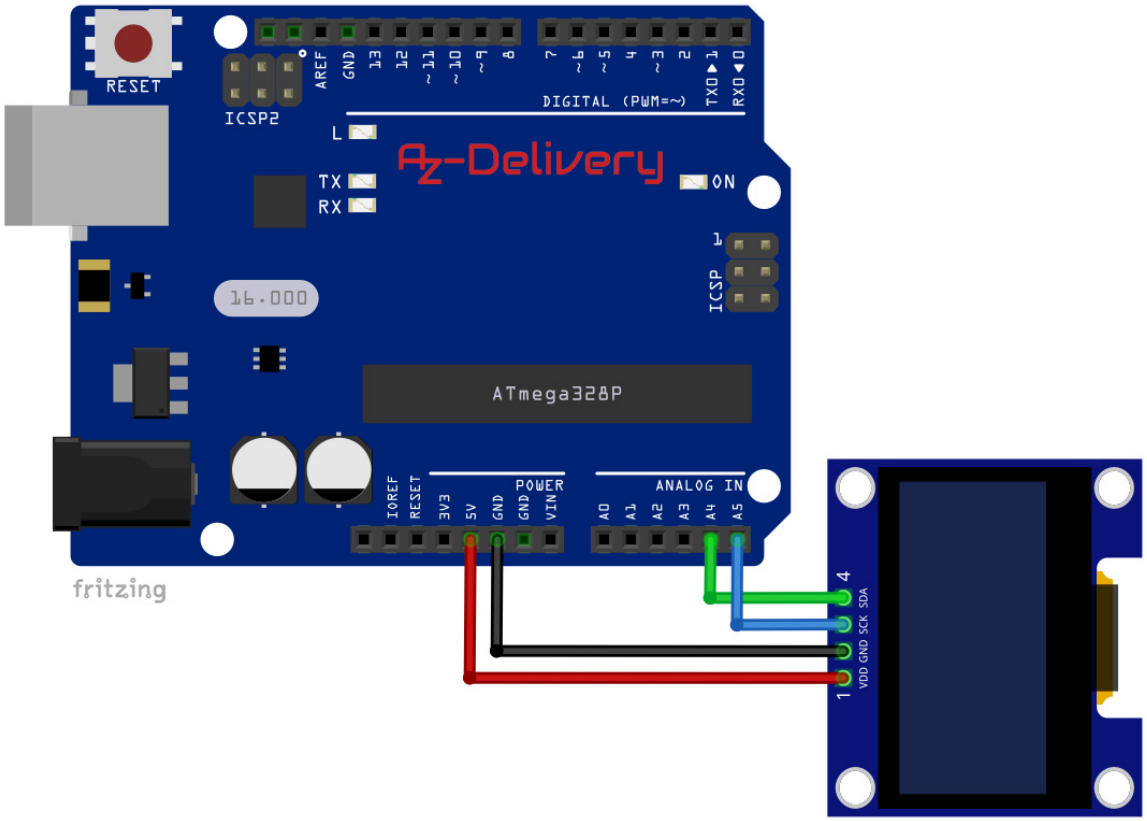


L'écran dispose d'un régulateur de tension intégré. Les broches de l'écran OLED de 1,3” peuvent être connectées à une alimentation de 3,3V ou de 5V sans danger pour le capteur lui-même.

REMARQUE : Lorsque vous utilisez Raspberry Pi, l'alimentation doit être tirée d'une broche de 3,3V uniquement.

Connexion de l'écran avec l'Atmega328P Board

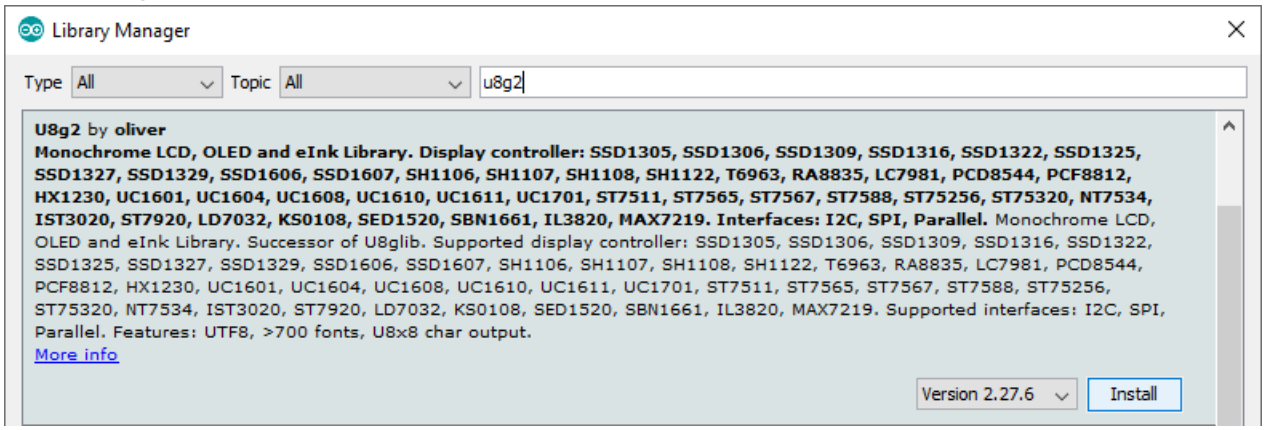
Connectez l'écran OLED de 1,3” avec l' Atmega328P Board comme indiqué sur le schéma de connexion suivant :



Broche d'écran	Broche	Couleur du fil
SDA	A4	Câble vert
SCK	A5	Câble bleu
GND	GND	Câble noir
VCC	5V	Câble rouge

Bibliothèque pour l'IDE Arduino

Pour utiliser l'écran avec un Atmega328P Board, il est recommandé de télécharger une bibliothèque externe pour celui-ci. La bibliothèque qui va être utilisée s'appelle "U8g2". Pour la télécharger et l'installer, ouvrez Arduino IDE et allez dans : Outils > Gérer les bibliothèques. Lorsqu'une nouvelle fenêtre s'ouvre, tapez "u8g2" dans le champ de recherche et installez la bibliothèque "U8g2" réalisée par "oliver", comme le montre l'image suivante :



Plusieurs exemples de sketchs sont fournis avec la bibliothèque, pour en ouvrir un, allez dans :
Fichier > Exemples > U8g2 > full_buffer > GraphicsTest
Avec cet exemple de sketch, vous pouvez tester votre écran. Cependant, le code utilisé dans l'exemple est assez complexe. Le sketch est modifié pour rendre le code plus facile à comprendre pour les débutants.

Exemple de croquis

```
#include <U8g2lib.h>
#include <Wire.h>
#define time_delay 2000
U8G2_SSD1306_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, U8X8_PIN_NONE);

const char COPYRIGHT_SYMBOL[] = {0xa9, '\0'};
void u8g2_prepare() {
  u8g2.setFont(u8g2_font_6x10_tf);
  u8g2.setFontRefHeightExtendedText();
  u8g2.setDrawColor(1);
  u8g2.setFontPosTop();
  u8g2.setFontDirection(0);
}
void u8g2_box_frame() {
  u8g2.drawStr(0, 0, "drawBox");
  u8g2.drawBox(5, 10, 20, 10);
  u8g2.drawStr(60, 0, "drawFrame");
  u8g2.drawFrame(65, 10, 20, 10);
}
void u8g2_r_frame_box() {
  u8g2.drawStr(0, 0, "drawRFrame");
  u8g2.drawRFrame(5, 10, 40, 15, 3);
  u8g2.drawStr(70, 0, "drawRBox");
  u8g2.drawRBox(70, 10, 25, 15, 3);
}
void u8g2_disc_circle() {
  u8g2.drawStr(0, 0, "drawDisc");
  u8g2.drawDisc(10, 18, 9);
  u8g2.drawStr(60, 0, "drawCircle");
  u8g2.drawCircle(70, 18, 9);
}
```

```
void u8g2_string_orientation() {
  u8g2.setFontDirection(0);
  u8g2.drawStr(5, 15, "0");
  u8g2.setFontDirection(3);
  u8g2.drawStr(40, 25, "90");
  u8g2.setFontDirection(2);
  u8g2.drawStr(75, 15, "180");
  u8g2.setFontDirection(1);
  u8g2.drawStr(100, 10, "270");
}
void u8g2_line() {
  u8g2.drawStr(0, 0, "drawLine");
  u8g2.drawLine(7, 20, 77, 32);
}
void u8g2_triangle() {
  u8g2.drawStr(0, 0, "drawTriangle");
  u8g2.drawTriangle(14, 20, 45, 30, 10, 32);
}
void u8g2_unicode() {
  u8g2.drawStr(0, 0, "Unicode");
  u8g2.setFont(u8g2_font_unifont_t_symbols);
  u8g2.setFontPosTop();
  u8g2.setFontDirection(0);
  u8g2.drawUTF8(10, 20, "☼");
  u8g2.drawUTF8(30, 20, "☁");
  u8g2.drawUTF8(50, 20, "☂");
  u8g2.drawUTF8(70, 20, "☔");
  u8g2.drawUTF8(95, 20, COPYRIGHT_SYMBOL); //COPYRIGHT SIMBOL
  u8g2.drawUTF8(115, 15, "\xb0"); // DEGREE SYMBOL
}
```

```
#define image_width 128
#define image_height 21
static const unsigned char image_bits[] U8X8_PROGMEM = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x06, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x1f, 0x00, 0x00,
  0xfc, 0x1f, 0x00, 0x00, 0x06, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0xfe, 0x1f, 0x00, 0x00, 0xfc, 0x7f, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x07, 0x18, 0x00, 0x00, 0x0c, 0x60, 0x00, 0x00,
  0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x18, 0x00, 0x00,
  0x0c, 0xc0, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x03, 0x18, 0x00, 0x00, 0x0c, 0xc0, 0xf0, 0x1f, 0x06, 0x63, 0x80, 0xf1,
  0x1f, 0xfc, 0x33, 0xc0, 0x03, 0x18, 0x00, 0x00, 0x0c, 0xc0, 0xf8, 0x3f,
  0x06, 0x63, 0xc0, 0xf9, 0x3f, 0xfe, 0x33, 0xc0, 0x03, 0x18, 0x00, 0x00,
  0x0c, 0xc0, 0x18, 0x30, 0x06, 0x63, 0xc0, 0x18, 0x30, 0x06, 0x30, 0xc0,
  0xff, 0xff, 0xdf, 0xff, 0x0c, 0xc0, 0x18, 0x30, 0x06, 0x63, 0xe0, 0x18,
  0x30, 0x06, 0x30, 0xc0, 0xff, 0xff, 0xdf, 0xff, 0x0c, 0xc0, 0x98, 0x3f,
  0x06, 0x63, 0x60, 0x98, 0x3f, 0x06, 0x30, 0xc0, 0x03, 0x18, 0x0c, 0x00,
  0x0c, 0xc0, 0x98, 0x1f, 0x06, 0x63, 0x70, 0x98, 0x1f, 0x06, 0x30, 0xc0,
  0x03, 0x18, 0x06, 0x00, 0x0c, 0xc0, 0x18, 0x00, 0x06, 0x63, 0x38, 0x18,
  0x00, 0x06, 0x30, 0xc0, 0x03, 0x18, 0x03, 0x00, 0x0c, 0xe0, 0x18, 0x00,
  0x06, 0x63, 0x1c, 0x18, 0x00, 0x06, 0x30, 0xc0, 0x00, 0x80, 0x01, 0x00,
  0xfc, 0x7f, 0xf8, 0x07, 0x1e, 0xe3, 0x0f, 0xf8, 0x07, 0x06, 0xf0, 0xcf,
  0x00, 0xc0, 0x00, 0x00, 0xfc, 0x3f, 0xf0, 0x07, 0x1c, 0xe3, 0x07, 0xf0,
  0x07, 0x06, 0xe0, 0xcf, 0x00, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0, 0x00, 0x30, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0,
  0x00, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0xe0, 0x00, 0xfc, 0x1f, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7f, 0x00, 0xfc, 0x1f, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3f };
```

```
void u8g2_bitmap() {
  u8g2.drawXBMP(0, 5, image_width, image_height, image_bits);
}
void setup(void) {
  u8g2.begin();
  u8g2_prepare();
}
float i = 0.0;
void loop(void) {
  u8g2.clearBuffer();
  u8g2_prepare();
  u8g2_box_frame();
  u8g2.sendBuffer();
  delay(time_delay);

  u8g2.clearBuffer();
  u8g2_disc_circle();
  u8g2.sendBuffer();
  delay(time_delay);

  u8g2.clearBuffer();
  u8g2_r_frame_box();
  u8g2.sendBuffer();
  delay(time_delay);

  u8g2.clearBuffer();
  u8g2_prepare();
  u8g2_string_orientation();
  u8g2.sendBuffer();
  delay(time_delay);

  u8g2.clearBuffer();
  u8g2_line();
  u8g2.sendBuffer();
  delay(time_delay);
```

```
  // one tab
  u8g2.clearBuffer();
  u8g2_triangle();
  u8g2.sendBuffer();
  delay(time_delay);

  u8g2.clearBuffer();
  u8g2_prepare();
  u8g2_unicode();
  u8g2.sendBuffer();
  delay(time_delay);

  u8g2.clearBuffer();
  u8g2_bitmap();
  u8g2.sendBuffer();
  delay(time_delay);

  u8g2.clearBuffer();
  u8g2.setCursor(0, 0);
  u8g2.print(i);
  i = i + 1.5;
  u8g2.sendBuffer();
  delay(time_delay);
}
```

Tout d'abord, deux bibliothèques, *U8g2lib* et *Wire* sont importées. Ensuite, un objet appelé u8g2 est créé avec la ligne de code suivante :

```
U8G2_SSD1306_128X32_UNIVISION_F_HW_I2C u8g2(U8G2_R0, U8X8_PIN_NONE);
```

L'objet créé représente l'écran lui-même et il est utilisé pour contrôler l'écran. La bibliothèque U8g2 peut être utilisée pour de nombreux autres écrans OLED, c'est pourquoi il y a de nombreux constructeurs dans les exemples de sketchs issus de la bibliothèque.

Ensuite, une fonction appelée *u8g2_prepare()* est créée, qui n'a pas d'arguments et ne renvoie aucune valeur. Les cinq fonctions de la bibliothèque u8g2 sont utilisées.

La première fonction s'appelle *setFont()*, elle a un argument et ne renvoie aucune valeur. L'argument représente la fonte u8g2. Suivez le lien pour voir la liste des fontes disponibles :
https://github.com/olikraus/u8g2/wiki/fntlist8x8

La deuxième fonction s'appelle *setFontRefHeightExtendedText()*, elle n'a pas d'arguments et ne renvoie aucune valeur. Elle est utilisée pour dessiner les caractères à l'écran. Pour une explication plus détaillée, suivez le lien :
https://github.com/olikraus/u8g2/wiki/u8g2reference#setfontrefheightextendedtext

La troisième fonction est appelée *setDrawColor()*, qui a un argument et ne renvoie aucune valeur. La valeur de l'argument est un nombre entier qui représente un indice de couleur pour toutes les fonctions de dessin. Les procédures de dessin de polices utilisent cet argument pour définir la couleur de premier plan. La valeur par défaut est 1. Si elle est fixée à 0, l'espace autour du caractère est éclairé, mais le caractère ne l'est pas. La valeur 2 de l'argument peut également être utilisée, mais il n'y a pas de différence avec 0.

La quatrième fonction s'appelle *setFontPosTop()*, elle n'a pas d'argument et ne renvoie aucune valeur. Cette fonction contrôle la position du caractère dans une ligne de texte. Il existe plusieurs versions de cette fonction. La première est *setFontPosBaseLine()*, la deuxième est *setFontPosCenter()*, et la troisième est *setFontPosBottom()*. Leur but est de changer la position des caractères sur une ligne.

La cinquième fonction est appelée *setFontDirection()*, qui a un argument et ne renvoie aucune valeur. L'argument est un nombre entier qui représente la direction du texte. La valeur est un nombre entier compris entre 0 et 3 (*0 = 0°*, *1 = 90°*, *2 = 180°* et *3 = 270°*).

La fonction appelée *drawStr()* possède trois arguments et ne renvoie aucune valeur. Elle est utilisée pour afficher une chaîne de caractères constante à l'écran. Les deux premiers arguments représentent les positions *X* et *Y* du curseur, où le texte est affiché. Le troisième argument représente le texte lui-même, une valeur de chaîne de caractères. Les fonctions qui configurent la mise en page du texte doivent être utilisées avant la fonction *drawStr()*, sinon la fonction *drawStr()* utilisera les paramètres par défaut pour la fonte, la taille et la disposition générale du texte.

Pour afficher les formes, il faut utiliser des fonctions de bibliothèque spécifiques pour chaque forme :

La fonction appelée *drawFrame()* possède quatre arguments et ne renvoie aucune valeur. Elle est utilisée pour afficher le cadre, un rectangle vide. Les deux premiers arguments représentent les positions *X* et *Y* du coin supérieur gauche du cadre. Le troisième argument représente la largeur du cadre et le quatrième argument représente la hauteur du cadre.

La fonction appelée *drawRFrame()* possède cinq arguments et ne renvoie aucune valeur. Elle est utilisée pour afficher un cadre aux coins arrondis. Les deux premiers arguments représentent les positions *X* et *Y* du coin supérieur gauche du cadre. Les deux seconds arguments représentent la largeur et la hauteur du cadre et le cinquième argument représente le rayon du coin.

La fonction appelée *drawBox()* possède quatre arguments et ne renvoie aucune valeur. Elle est utilisée pour afficher un rectangle rempli. Les deux premiers arguments représentent les positions *X* et *Y* du coin supérieur gauche du rectangle. Les deux autres arguments représentent la largeur et la hauteur du rectangle.

La fonction appelée *drawRBox()* possède cinq arguments et ne renvoie aucune valeur. Elle est utilisée pour afficher un rectangle rempli avec des bords arrondis. Les deux premiers arguments représentent les positions *X* et *Y* du coin supérieur gauche du rectangle. Les deux seconds arguments représentent la largeur et la hauteur du rectangle et le cinquième argument représente le rayon du coin.

La fonction appelée *drawCircle()* possède trois arguments et ne renvoie aucune valeur. Elle est utilisée pour afficher un cercle. Les deux premiers arguments représentent les positions *X* et *Y* du point central du cercle. Le troisième argument représente le rayon du cercle.

La fonction appelée *drawDisc()* possède trois arguments et ne renvoie aucune valeur. Elle est utilisée pour afficher un disque. Les deux premiers arguments représentent les positions *X* et *Y* du point central du disque. Le troisième argument représente le rayon du disque.

La fonction appelée *drawTriangle()* possède six arguments et ne renvoie aucune valeur. Elle est utilisée pour afficher un triangle rempli. Les deux premiers arguments représentent les positions X et Y du premier point d'angle du triangle. Les deux seconds arguments représentent les positions *X* et *Y* du second point d'angle du triangle. Les deux derniers arguments représentent les positions *X* et *Y* du dernier point d'angle du triangle.

La fonction appelée drawLine() possède quatre arguments et ne renvoie aucune valeur. Elle est utilisée pour afficher une ligne. Les deux premiers arguments représentent les positions *X* et *Y* du point de départ de la ligne. Les deux autres arguments représentent les positions *X* et *Y* du point final de la ligne.

La fonction appelée *drawUTF8()* possède trois arguments et renvoie une valeur. Elle est utilisée pour afficher un texte, la valeur d'une chaîne de caractères qui peut contenir un caractère encodé en tant que caractère Unicode. Les deux premiers arguments représentent les positions *X* et *Y* du curseur et le troisième représente le texte lui-même. Les caractères Unicode peuvent être affichés de plusieurs façons. La première consiste à copier et coller le caractère existant dans le sketch, comme dans la ligne de code suivante : u8g2.drawUTF8(50, 20, "☂") ;

La seconde consiste à créer un tableau de chars, qui a deux valeurs : la première valeur est un numéro hexadécimal du caractère Unicode et la seconde valeur est un caractère nul. Ceci peut être fait en utilisant le tableau de caractères appelé COPYRIGHT_SYMBOL, comme dans les lignes de code suivantes :

```
const char COPYRIGHT_SYMBOL[] = {0xa9, '\0'};
u8g2.drawUTF8(95,   20,   COPYRIGHT_SYMBOL);   //COPYRIGHT
SYMBOL
```

La troisième façon d'utiliser la fonction est d'utiliser un nombre hexadécimal pour le caractère lui-même, comme dans la ligne de code suivante :

```
u8g2.drawUTF8(115, 15, "\xb0"); // DEGREE SYMBOL
```

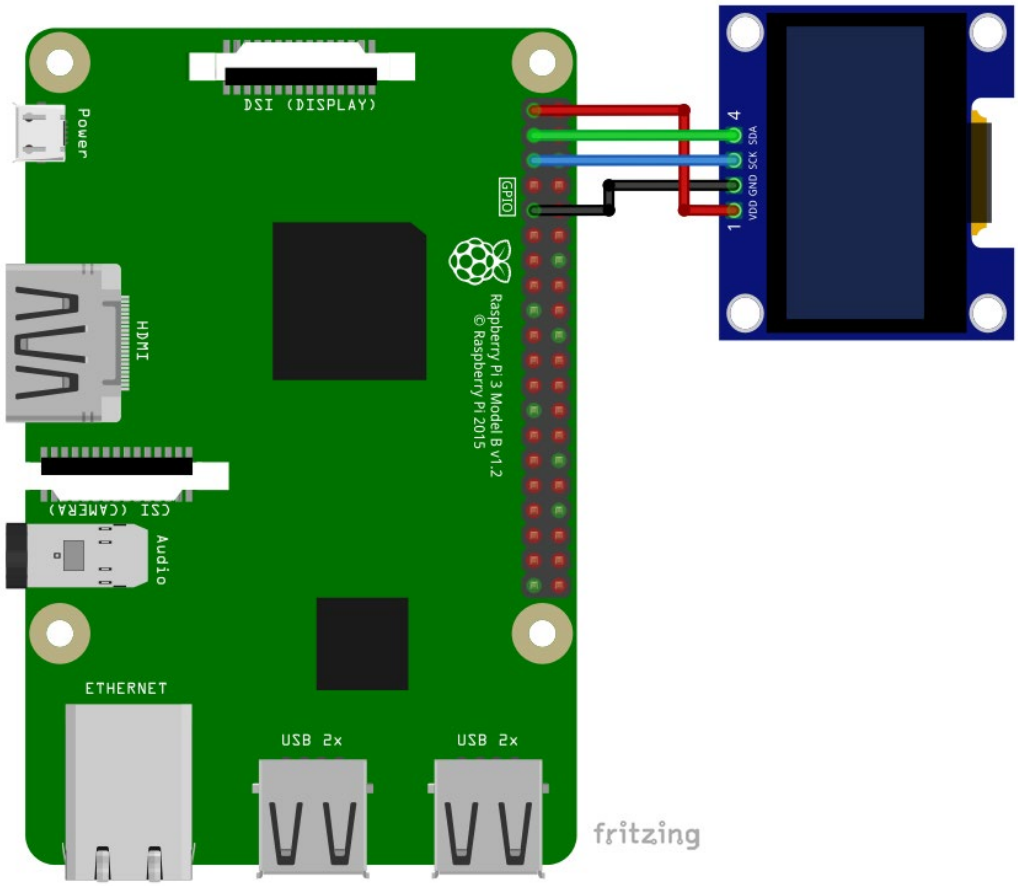
La fonction renvoie une valeur, un nombre entier qui représente la largeur du texte (chaîne de caractères).

Pour afficher quelque chose à l'écran, le tampon de données de l'écran doit être effacé, puis une nouvelle valeur doit être définie pour le tampon de données (une image) et enfin la nouvelle valeur doit être envoyée à l'écran. De cette façon, une nouvelle image sera affichée à l'écran. Afin de rendre ce changement visible, la fonction delay() doit être utilisée pour retarder le prochain changement du tampon de données, comme dans les lignes de code suivantes :

```
u8g2.clearBuffer();
u8g2_bitmap(); // setting the data buffer
u8g2.sendBuffer();
delay(time_delay);
```

Connexion de l'écran avec Raspberry Pi

Connectez l'écran avec le Raspberry Pi comme indiqué sur le schéma de connexion suivant :

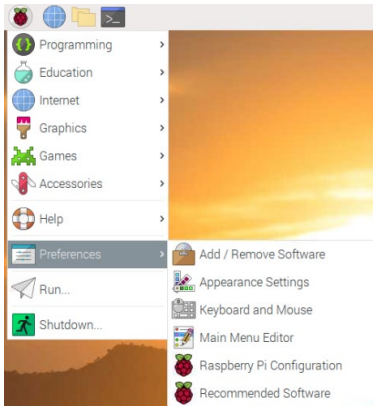


Broche d'écran	Raspberry Pi pin	Broche Physique	Couleur du câble
SDA	GPIO2	3	**Câble vert**
SCL	GPIO3	5	**Câble bleu**
GND	GND	9	**Câble noir**
VCC	3V3	1	**Câble rouge**

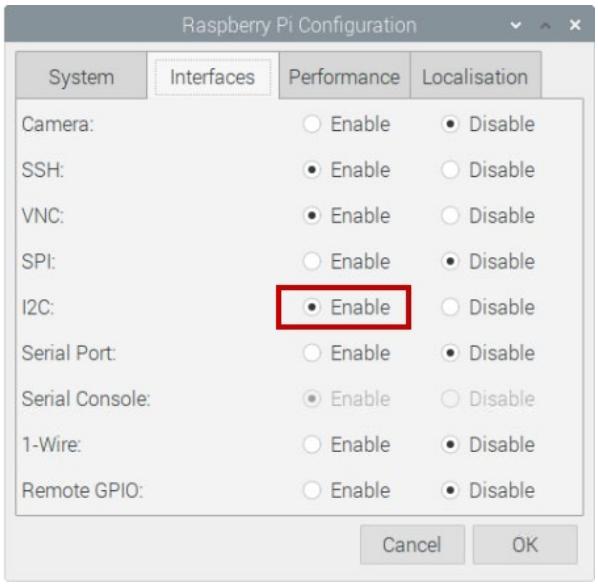
Activation de l'interface I2C

Afin d'utiliser le module avec Raspberry Pi, l'interface I2C doit être activée. Ouvrez le menu suivant :

Application Menu > Preferences > Raspberry Pi Configuration



Dans la nouvelle fenêtre, sous l'onglet Interfaces, activez le bouton radio I2C, comme sur l'image suivante :



Bibliothèques et outils pour Python

Pour utiliser un écran OLED de 1,3” avec un Raspberry Pi, il est recommandé de télécharger une bibliothèque externe pour celui-ci. La bibliothèque à utiliser s'appelle "luma.oled", mais avant cela, quelques outils pour Python doivent être installés.

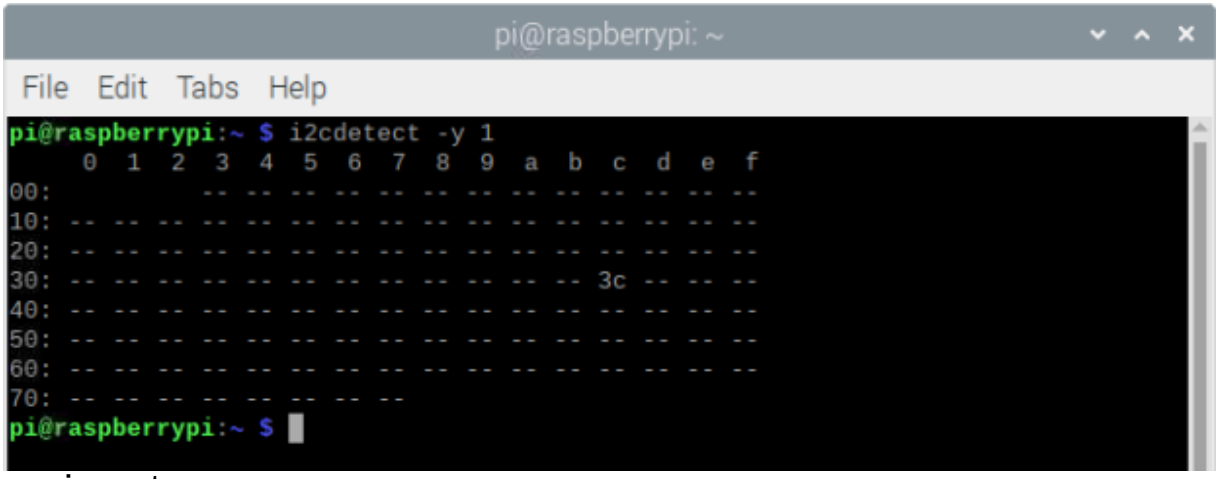
Ouvrez le terminal et exécutez la commande suivante :

sudo apt install i2c-tools python3-dev python3-pip libfreetype6-dev libjpeg-dev build-essential libopenjp2-7 libtiff5 git

Ensuite, l'adresse I2C de l'écran doit être détectée. Dans la fenêtre du terminal, exécutez la commande suivante :

i2c detect -y 1

Le résultat devrait ressembler à celui de l'image suivante :



où 0x3c est l'adresse I2C de l'écran.

Pour installer la bibliothèque luma.oled, ouvrez le terminal et exécutez la commande suivante :

sudo -H pip3 install –upgrade luma.oled

La dernière chose à faire est de télécharger les exemples de script depuis GitHub. Dans la fenêtre du terminal, exécutez la commande suivante :

git clone https://github.com/rm-hull/luma.examples.git

Ensuite, (dans le terminal) changez de répertoire pour le répertoire luma.examples, avec la commande suivante : **cd luma.examples**

et pour l'installer, exécutez la commande suivante :

sudo -H pip3 install -e .

(avec le point à la fin)

Python script

Le code du script suivant est un code modifié de deux scripts :

/luma.examples/demo.py and */luma.examples/demo_opts.py*.

```
import time
import sys
from luma.core.interface.serial import i2c
from luma.oled.device import sh1106
from luma.core.render import canvas
from PIL import ImageFont

font_path = 'ChiKareGo.ttf'

def changing_var(device):
    size = 40
    nf = ImageFont.truetype(font_path, size) # new font

    for i in range(100):
        with canvas(device) as draw:
            draw.text((28, 7), 'Changing var.', fill=1)
            if i < 10:

draw.text((50,22),'0{}'.format(str(i)),font=nf,fill=1)
            else:
               draw.text((50, 22), str(i), font=nf, fill=1)

            time.sleep(0.001)
```

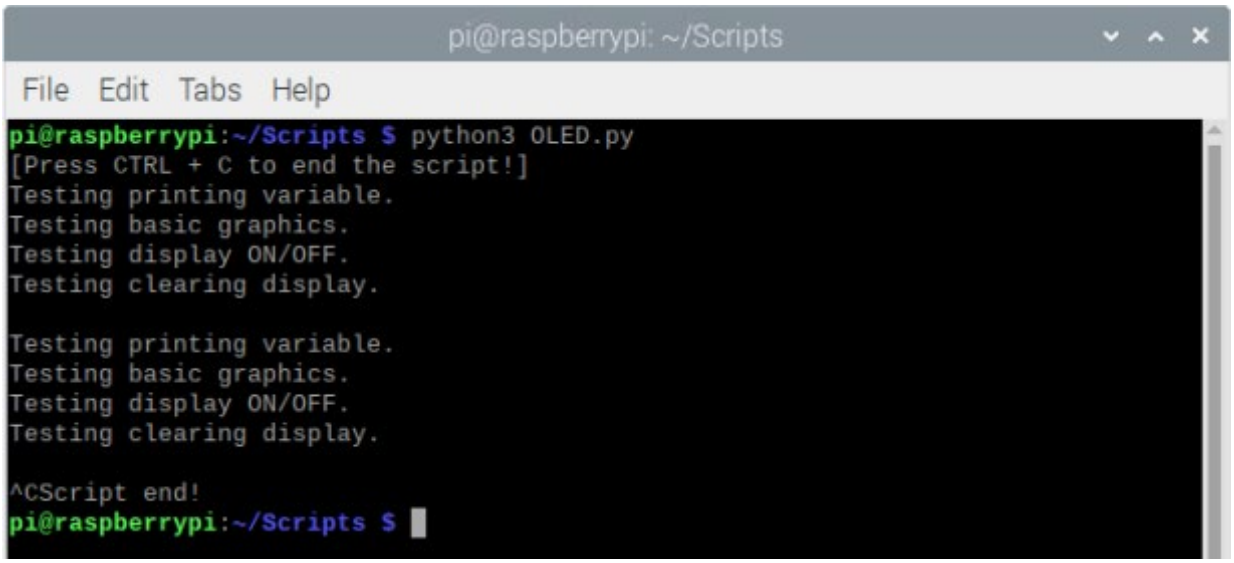
```
def primitives(device):
    with canvas(device) as draw:
        # Draw a rectangle.
        draw.rectangle((4, 4, 40, 10), outline=1, fill=0)

        # Draw an ellipse.
        draw.ellipse((4, 20, 18, 34), outline=1, fill=1)

        # Draw a triangle.

draw.polygon([(10,44),(40,20),(40,44)],outline=1,fill=0)

        # Draw an X.
        draw.line((4, 48, 126, 62), fill=1)
        draw.line((4, 62, 126, 48), fill=1)

        # Write two lines of text.
        draw.text((45, 20), 'AZ-Delivery', fill=1)

        size = 10
        nf = ImageFont.truetype(font_path, size)
        draw.text((45, 4), 'AZ-Delivery', font=nf, fill=1)
```

```
try:
    serial = i2c(port=1, address=0x3c)
    device = sh1106(serial, rotate=0, width=128, height=64)

    print('[Press CTRL + C to end the script!]')
    while(True):
        print('Testing printing variable.')
        changing_var(device)
        time.sleep(2)

        print('Testing basic graphics.')
        primitives(device)
        time.sleep(3)

        print('Testing display ON/OFF.')
        for _ in range(5):
           time.sleep(0.5)
           device.hide()
           time.sleep(0.5)
           device.show()

        print('Testing clearing display.\n')
        device.clear()
        time.sleep(2)

except KeyboardInterrupt:
    print('Script end!')
```

Enregistrez le script sous le nom "OLED.py". Pour exécuter le script, ouvrez le terminal dans le répertoire où le script est enregistré et exécutez la commande suivante :

`python3 OLED.py`

Le résultat devrait ressembler à celui de l'image suivante :



Pour arrêter le script, appuyez sur 'CTRL + C' sur le clavier.

Le script commence par l'importation de bibliothèques et de fonctions.

Ensuite, le chemin est enregistré dans la fonte spéciale dans la variable *font_path*. La fonte utilisée est appelée ChiKareGo.ttf, qui peut être trouvée dans : *luma.examples* > *examples* > *fonts* > *ChiKareGo.ttf*

Le fichier de fonte doit être copié dans le même répertoire que celui où le script est enregistré.

Ensuite, deux fonctions sont créées, la première est appelée *changing_var()* et la seconde est appelée *primitives()*.

La fonction *changing_var()* est utilisée pour afficher une variable qui change d'état toutes les millisecondes. La fonction accepte un argument et ne renvoie aucune valeur. L'argument est appelé argument de périphérique, ce qui sera expliqué plus loin dans le texte. Au début de la fonction *changing_var()*, la variable size est définie et initialisée avec le nombre 40. Cette valeur représente la taille de la fonte. Ensuite, une variable appelée nf (new_font) est créée, qui représente la police utilisée. Les variables size et font_path sont utilisées pour créer la variable nf. Ensuite, la for boucle est créée pour parcourir en boucle les cent premiers nombres entiers. Ces valeurs sont utilisées comme valeurs pour la variable qui sera affichée à l'écran. Dans la for boucle, on crée l'image qui sera affichée à l'écran, avec la ligne de code suivante :
avec canvas(device) comme draw :
où un objet appelé *sdraw* est créé. L'objet *draw* représente l'image elle-même.

Pour afficher le texte, on utilise la fonction *text()*. La fonction *text()* accepte quatre arguments, dont l'un est facultatif, et ne renvoie aucune valeur. Le premier argument est un tuple avec deux éléments, où les éléments représentent les positions X et Y du curseur. Le deuxième argument représente le texte lui-même, une valeur de type chaîne. Le troisième argument est l'*argument de remplissage*. Le quatrième argument est facultatif, il est appelé l'argument de fonte. Si cet argument n'est pas utilisé, la fonte sera définie par la portée et la taille par défaut. Mais si la *variable nf* est stockée dans l'argument de la source, la fonte désirée peut être utilisée et la taille de la fonte modifiée.

L'argument fill représente la couleur du texte. Si le *fill* = 0, la couleur sera noire (rien ne sera affiché à l'écran) et si le *fill* est égal à une autre valeur, par exemple *fill* = 1, la couleur du texte sera blanche (car les écrans OLED n'ont que des couleurs noires et blanches). Cette bibliothèque peut également être utilisée pour de nombreux autres écrans, ainsi à l'intérieur de l'argument fill, toute autre couleur peut être stockée, ce qui n'est pas possible pour un écran OLED de 1,3 pouces.

Le reste du code à l'intérieur de la boucle for est un algorithme qui affiche du texte et une variable avec des nombres à deux chiffres allant de 00 à 99.

La deuxième fonction s'appelle *primitives()* et elle est utilisée pour dessiner de nombreuses formes à l'écran. Elle n'accepte qu'un seul argument, celui du périphérique, et ne renvoie aucune valeur. Au début de la fonction, l'objet image *draw* est créé. Cet objet est utilisé pour dessiner de nombreuses formes.

Pour dessiner le rectangle, on utilise la fonction *rectangle()*. Cette fonction accepte trois arguments et ne renvoie aucune valeur. Le premier argument est un tuple de quatre éléments, où les deux premiers éléments représentent les positions *X* et *Y* du coin supérieur droit du rectangle. Le troisième et le quatrième élément représentent les positions *X* et *Y* du coin inférieur droit du rectangle. Le deuxième argument est l'argument de contour et le troisième argument est l'argument de remplissage.

La valeur de la position X commence du côté gauche de l'écran (valeur 0) et se termine du côté droit de l'écran (valeur 127). La valeur de la position Y commence en haut de l'écran (valeur de 0) et se termine en bas de l'écran (valeur de 63).

L'argument *outline* représente la couleur du bord de la forme et l'argument *fill* représente la couleur de la forme elle-même. Comme les écrans OLED sont utilisés, les couleurs sont le noir et le blanc, noir - le pixel est éteint *(OFF)*, blanc - le pixel est allumé *(ON)*. Lorsque la valeur zéro est enregistrée dans l'argument outline ou *fill*, cela signifie qu'il s'agit d'une couleur noire. Lorsque toute autre valeur supérieure à zéro est enregistrée, par exemple *1*, cela représente une couleur blanche.

Pour dessiner une ellipse ou des cercles, on utilise la fonction *ellipse()*. La fonction accepte trois arguments et ne renvoie aucune valeur. Le premier argument est un tuple de quatre éléments. Les deux premiers éléments représentent les positions *X* et *Y* du coin supérieur gauche du rectangle qui contient l'ellipse. Les deux autres éléments représentent les positions *X* et *Y* du coin inférieur droit du rectangle qui contient l'ellipse. Le deuxième argument est l'argument de contour et le troisième argument est l'argument de *remplissage*.

Pour dessiner un polygone, on utilise la fonction *polygon()*. Un polygone est un triangle, un rectangle, ou toute autre forme avec 3 coins ou plus. La fonction accepte trois arguments. Le premier argument est une liste de trois tuples ou plus. Les *tuples* ont deux éléments, qui représentent les positions *X* et *Y* du point d'angle de la forme. Le nombre de *tuples* dans la liste est arbitraire, trois tuples ou plus, ce qui dépend de la forme que l'on veut dessiner. Le deuxième argument est l'argument de contour et le troisième argument est l'argument de remplissage.

Pour dessiner une ligne, on utilise la fonction *line()*. Cette fonction accepte deux arguments et ne renvoie aucune valeur. Le premier argument est un tuple de quatre éléments, où les deux premiers éléments représentent les positions *X* et *Y* du point de départ d'une ligne et les deux seconds éléments représentent les positions *X* et *Y* du point d'arrivée d'une ligne. Le deuxième argument est l'argument de remplissage.

À la fin de la fonction primitives(), la fonction text() est utilisée avec et sans argument de fonte pour montrer la différence.

Après ces deux fonctions, un bloc de code *try-except* est créé. Dans le bloc de code *try*, deux objets et la boucle indéfinie sont créés.

Le premier objet est appelé serial et il est utilisé pour configurer l'interface I2C et l'adresse de l'écran. Pour initialiser cet objet, la ligne de code suivante est utilisée :

```
serial = i2c(port=1, address=0x3c)
```

Le deuxième objet est appelé périphérique. Cet objet représente la puce du pilote elle-même, et est utilisé pour contrôler la puce du pilote. Pour initialiser l'objet device, l'objet serial, les arguments width et height sont utilisés, dans la ligne de code suivante :

```
device = sh1106(serial, width=128, height=64)
```

Ensuite, une boucle indéfinie (while True :) est créée. À l'intérieur de la boucle indéfinie, deux fonctions sont exécutées et deux propriétés de l'écran sont testées. D'abord, la fonction *changing_var()* est exécutée, puis la fonction *primitives()*. Ensuite, *device.hide()* est utilisée pour éteindre l'écran et *device.show()* pour le rallumer. À la fin de la boucle indéfinie, *device.clear()* est utilisée pour effacer le tampon de données de l'écran, ce qui efface également l'image de l'écran.

Le bloc de code suivant est exécuté lorsqu'on appuie sur CTRL+C sur le clavier.

C'est ce qu'on appelle l'interruption du clavier. Lorsque le bloc de code *except* est exécuté, le message *"Script end !"* est imprimé dans le terminal.

Il est maintenant temps d'apprendre et de réaliser vos propres projets. Vous pouvez le faire à l'aide de nombreux exemples de scripts et autres tutoriels, que vous trouverez sur Internet.

Si vous recherchez des produits de haute qualité microélectronique et accessoires de haute qualité, AZ-Delivery Vertriebs GmbH est l'entreprise idéale pour vous les procurer. Vous recevrez de nombreux exemples d'application, des guides d'installation complets, des livres électroniques, des bibliothèques et l'assistance de nos experts techniques.

https://az-delivery.de

Amusez-vous !

Mentions légales

https://az-delivery.de/pages/about-us