Bienvenue !

Nous vous remercions d'avoir acheté notre écran AZ-Delivery OLED I2C de 0,96 pouce. 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 de 0,96 pouce comporte 128x64 pixels (LED). Pour contrôler ces LED, nous avons besoin d'un circuit de pilotage ou d'une puce. L'écran possède une puce pilote appelée SSD1306. La puce pilote possède une interface I2C pour communiquer avec le microcontrôleur principal. L'adresse I2C d'une puce pilote est prédéfinie, avec la valeur 0x3C.

L'écran OLED et la puce pilote SSD1306 fonctionnent dans la gamme 3.3V. Mais il existe un régulateur de tension 3.3V intégré, ce qui signifie que ces écrans peuvent fonctionner 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.

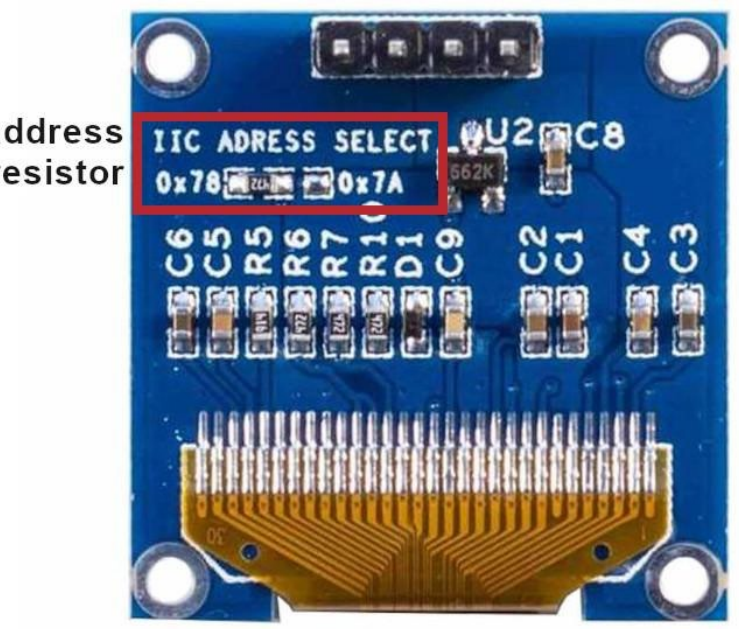
Caractéristiques

» Tension d'alimentation : de 3.3V à 5V DC
» Interface de communication : I2C
» Couleur du pixel : Blanc
» Température de fonctionnement : de -20 à 70 ℃
» Faible consommation d'énergie : moins de 11mA
» Dimensions : 28 x 33 x 4mm [1.1 x 1.3 x 0.13 in]

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

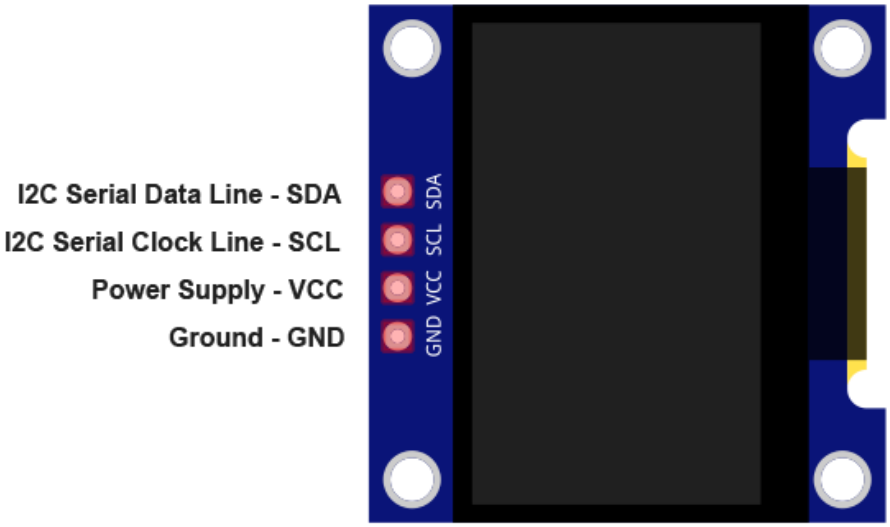
L'adresse I2C de l'écran OLED

Tous les écrans OLED proposés par AZ-Delivery ont la même adresse série, 0x3C. L'écran de 0,96 pouce vous offre la possibilité de ressouder une résistance à l'arrière de la carte de l'écran pour une autre adresse, 0x3D, mais ce n'est pas recommandé. Dans le cas où plusieurs écrans avec une interface I2C doivent être utilisés, il est recommandé d'utiliser un multiplexeur I2C ou simplement un écran plus grand.



Le pinout

L'écran OLED de 0,96 pouce possède quatre broches. Le brochage est montré sur l'image suivante :

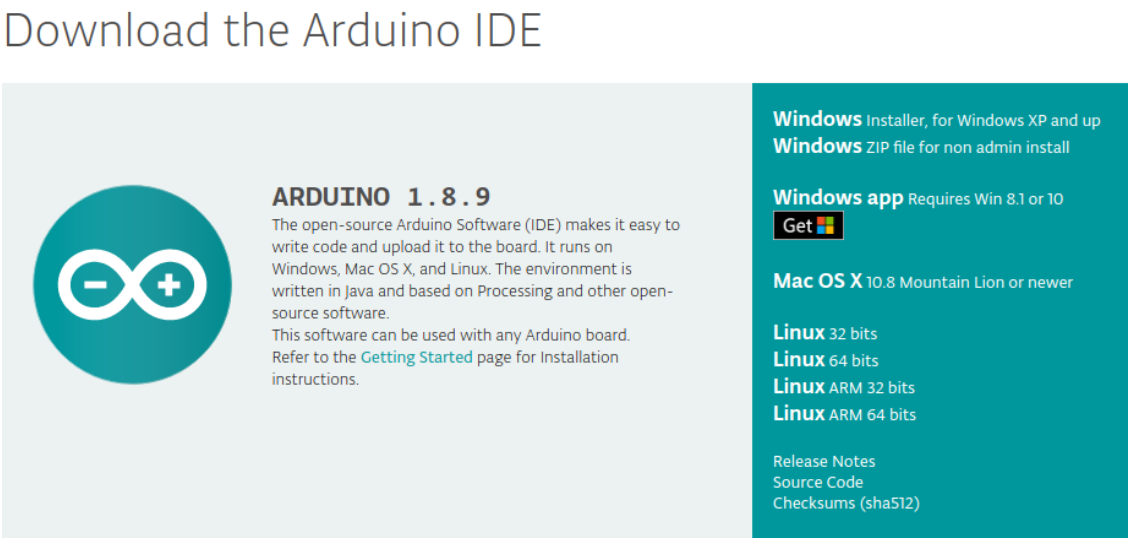


L'écran est doté d'un régulateur de tension de 3,3 V intégré. Les broches de l'écran OLED de 0,96 pouce peuvent être connectées à une alimentation de 3,3V ou de 5V sans danger pour l'écran lui-même.

NOTE : Lors de l'utilisation de Raspberry Pi, l'alimentation doit être uniquement tirée de la broche 3.3V.

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.

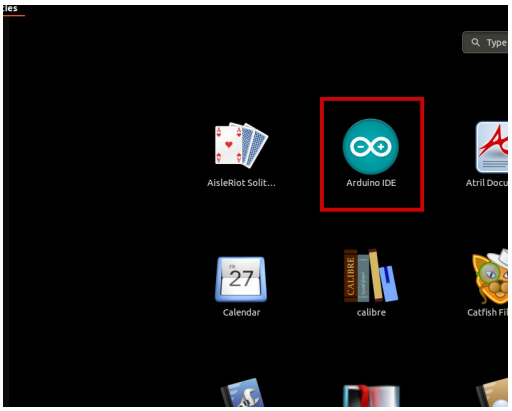
Pour les utilisateurs de Linux, téléchargez un fichier portant l'extension *.tar.xz*, qui doit être extrait. Lorsqu'il est extrait, allez dans le répertoire extrait et ouvrez le terminal dans ce répertoire. Deux scripts *.sh* doivent être exécutés, le premier appelé arduino-linux-setup.sh et le second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

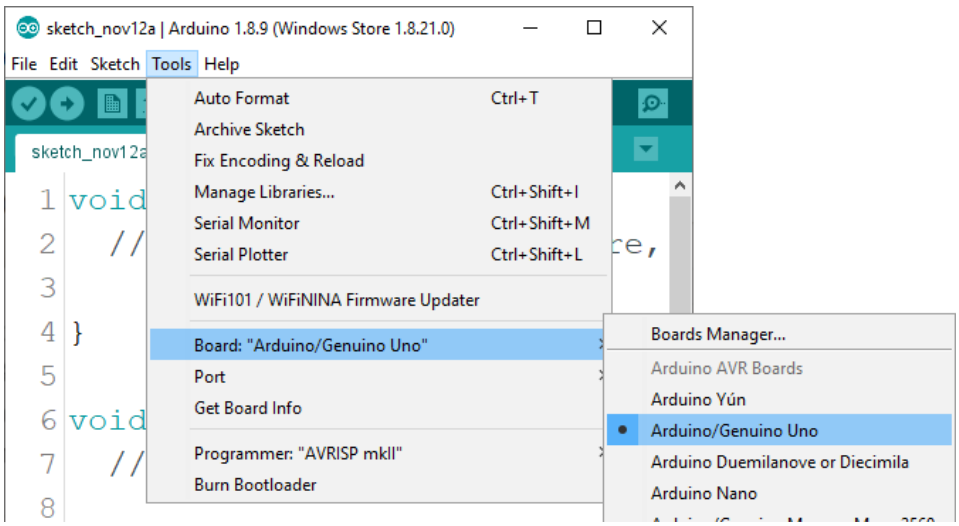
Le deuxième script appelé script *install.sh* doit être utilisé après l'installation du premier script. Exécutez la commande suivante dans le terminal (répertoire extrait) : **`sh install.sh`**

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



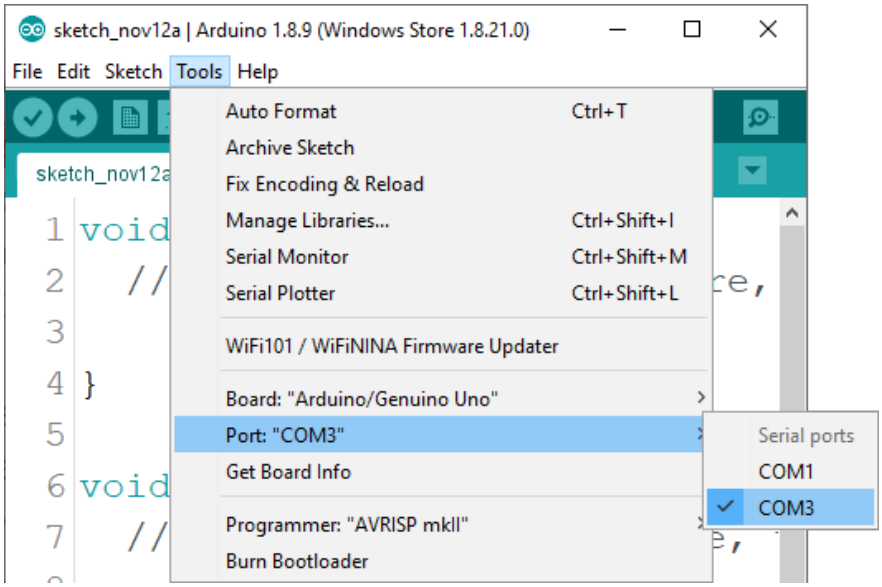
Presque tous les systèmes d'exploitation sont livrés avec un éditeur de texte préinstallé (par exemple, Windows est livré avec Notepad, Linux Ubuntu avec Gedit, Linux Raspbian avec Leafpad, etc.) Tous ces éditeurs de texte conviennent parfaitement à l'objectif de l'eBook.

La prochaine étape est de vérifier si votre PC peut détecter une carte microcontrôleur. Ouvrez l'IDE carte microcontrôleur fraîchement installé, et allez dans : *Tools > Board > {votre nom de carte ici}*
{votre nom de carte ici} devrait être l'Arduino/Genuino Uno, comme on peut le voir sur l'image suivante :



La porte à laquelle la carte du microcontrôleur est connectée doit être sélectionnée.
Allez-y : *Tools > Port > {le nom de la porte va ici}* et lorsque la carte microcontrôleur est connectée au port USB, le nom de la porte peut être vue 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.

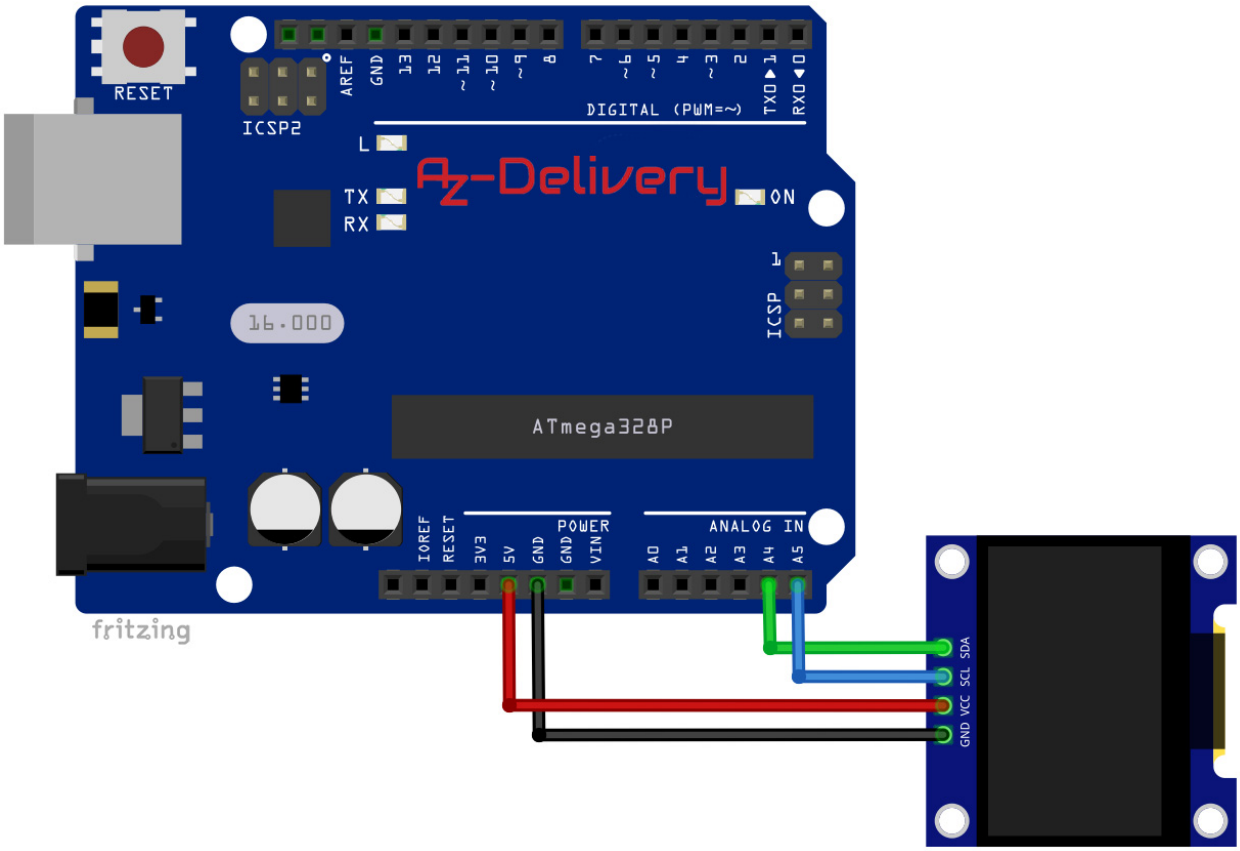
Comment configurer le Raspberry Pi et Python

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.

Connexion de l'écran avec le microcontrôleur

Connectez l'écran OLED de 0,96 pouce au microcontrôleur comme indiqué sur le schéma de connexion suivant :



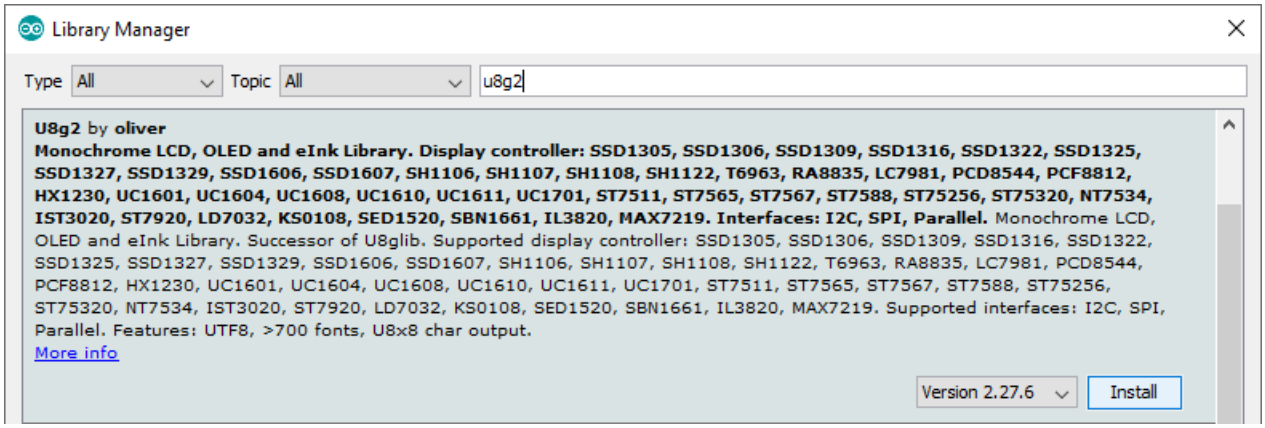
Broche d'écran	Broche MC	Couleur du Câble
SDA	A4	Câble bleu
SCL	A5	Câble vert
VCC	5V	Câble rouge
GND	GND	Câble noir

Bibliothèque pour l'IDE Arduino

Pour utiliser l'écran avec le microcontrôleur, il est recommandé de télécharger une bibliothèque externe pour celui-ci. La bibliothèque qui est utilisée dans cet eBook est appelée *U8g2*. Pour la télécharger et l'installer, ouvrez l'IDE Arduino et allez sur :

Outils > Gérer les bibliothèques.

Lorsqu'une nouvelle fenêtre s'ouvre, tapez *u8g2* dans la boîte de recherche et installez la bibliothèque U8g2 réalisée par Oliver, comme le montre l'image suivante :



Plusieurs exemples de croquis sont fournis avec la bibliothèque, pour en ouvrir un, allez à :

`File > Examples > U8g2 > full_buffer > GraphicsTest`

Avec cet exemple de sketch, l'écran peut être testé. Dans cet eBook, le code du sketch est modifié afin de créer une version du code plus accessible aux débutants.

Exemple de croquis

```
#include <U8g2lib.h>
#include <Wire.h>
#define time_delay 2000
U8G2_SSD1306_128X64_UNIVISION_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);
}
```

Au début du sketch, deux bibliothèques sont importées : U8g2lib et Wire.

Ensuite, un objet appelé u8g2 est créé, avec la ligne de code suivante :

```
U8G2_SSD1306_128X64_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 provenant de la bibliothèque.

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

La première fonction est appelée *setFont()* qui a un argument et ne renvoie aucune valeur. L'argument représente la fonte u8g2. La liste des fontes disponibles se trouve à l'adresse suivante *link*.

La deuxième fonction est appelée *setFontRefHeightExtendedText()* qui n'a pas d'arguments et ne renvoie aucune valeur. Elle est utilisée pour dessiner les caractères à l'écran. Une explication plus détaillée de cette fonction peut être trouvée sur le site suivant *link*

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 fontes utilisent cet argument pour définir la couleur de premier plan. La valeur par défaut est 1. Si elle est fixée à 0, alors l'espace autour du caractère est éclairé, et 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, appelée *setFontPosTop()*, n'a pas d'argument et ne renvoie aucune valeur. Cette fonction contrôle la position du caractère sur une ligne du texte. Il existe plusieurs versions de cette fonction. La première est *setFontPosBaseLine()*, la deuxième est setFontPosCenter(). La troisième est *setFontPosBottom()* et 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, où 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 la position X et Y du curseur, où le texte est affiché. Le troisième argument représente le texte lui-même, une valeur de chaîne constante. Il convient d'utiliser les fonctions qui définissent la disposition du texte avant d'utiliser la fonction drawStr(), sinon la fonction drawStr() utilise les paramètres par défaut pour la fonte, la taille et la disposition générale du texte.

Pour afficher les formes, on utilise des fonctions spécifiques pour chaque forme :

La fonction appelée drawFrame(), possède quatre arguments et ne renvoie aucune valeur. Elle est utilisée pour afficher un cadre, un rectangle vide. Les deux premiers arguments représentent la position *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 la position *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 la position *X* et *Y* du coin supérieur gauche du rectangle. Les deux autres arguments représentent la largeur et la hauteur du rectangle, respectivement.

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 la position *X* et *Y* du coin supérieur gauche du rectangle. Les deux seconds arguments représentent la largeur et la hauteur du rectangle, respectivement. 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 la position *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 la position *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 la position *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 de la chaîne de caractères qui peut contenir un caractère encodé en tant que caractère Unicode. Les deux premiers arguments représentent la position *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 deuxième est de créer un tableau de caractères, qui a deux valeurs : la première valeur est un numéro hexadécimal du caractère Unicode, et la deuxième valeur est un caractère nul ("\0"). Cela peut être fait en utilisant le tableau de caractères appelé COPYRIGHT_SYMBOL, comme dans les lignes suivantes du code :

```
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
```

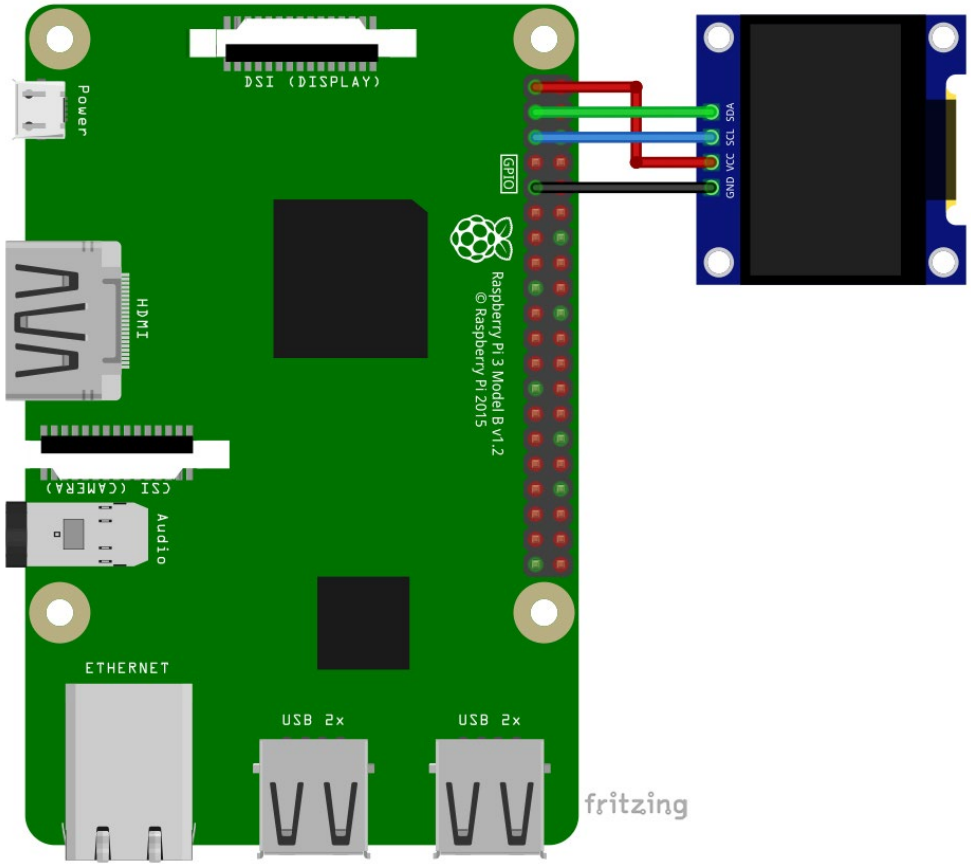
La fonction renvoie une valeur, un nombre entier qui représente la largeur du texte (chaîne de caractères).

Pour afficher quelque chose à l'écran, le serveur de données de l'écran doit d'abord être effacé, puis une nouvelle valeur est définie (une image) pour le serveur de données, puis une nouvelle valeur du serveur de données est envoyée à l'écran. De cette façon, une nouvelle image est affichée à l'écran. Afin de voir ce changement, la fonction delay() doit être utilisée pour décaler 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 :



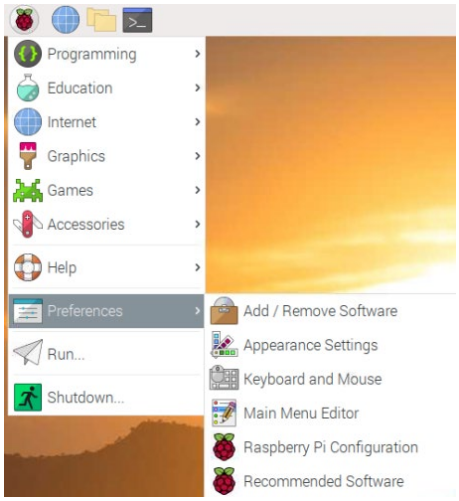
Broche d'écran	Broche Raspberry Pi	N° de la broche physique	Couleur du câble
SDA	GPIO2	3	**Câble Vert**
SCL	GPIO3	5	**Câble Bleu**
VCC	3V3	1	**Câble Rouge**
GND	GND	9	**Câble Noir**

Activation de l'interface I2C

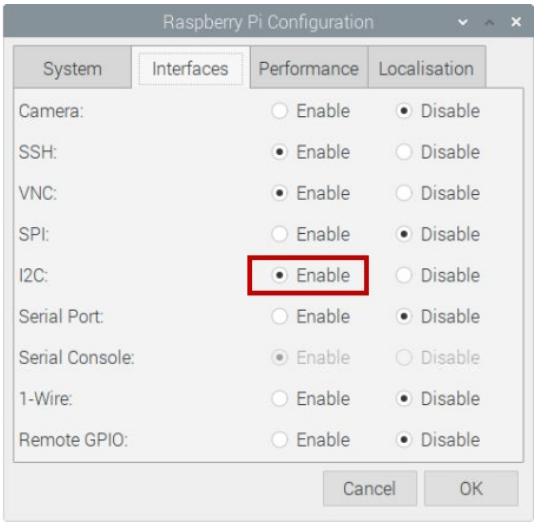
Afin d'utiliser l'écran avec Raspberry Pi, l'interface I2C doit être activée.

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

Afin d'utiliser l'écran avec le Raspberry Pi, il est recommandé de télécharger et d'installer une bibliothèque externe. La bibliothèque qui sera utilisée s'appelle *Adafruit_Python_SSD1306*.
Avant d'installer la bibliothèque principale, il faut d'abord installer plusieurs outils et bibliothèques.

D'abord, il fallait installer python-smbus et i2c-tools.
Ouvrez le terminal et exécutez les commandes suivantes :

```
sudo apt-get update
sudo apt-get install -y python-smbus i2c-tools
```

La liste suivante énumère les outils qui doivent être présents sur le système : *python3-dev*, *python-imaging*, *python3-pil*, *python3-pip*, *python3-setuptools*, *python3-rpi.gpio*

Si les outils de la liste ne sont pas présents, ils peuvent être installés en exécutant les commandes suivantes depuis la fenêtre du terminal :

```
sudo apt install -y python3-dev python-imaging python3-pil python3-pip python3-setuptools python3-rpi.gpio
```

Afin de télécharger la bibliothèque, le *git* doit être installé. Si le *git* n'est pas présent sur le système, l'installation peut être faite avec les commandes suivantes :

sudo apt install -y git

Le dépôt de la bibliothèque peut être cloné en exécutant la commande suivante :

git clone https://github.com/adafruit/Adafruit_Python_SSD1306.git

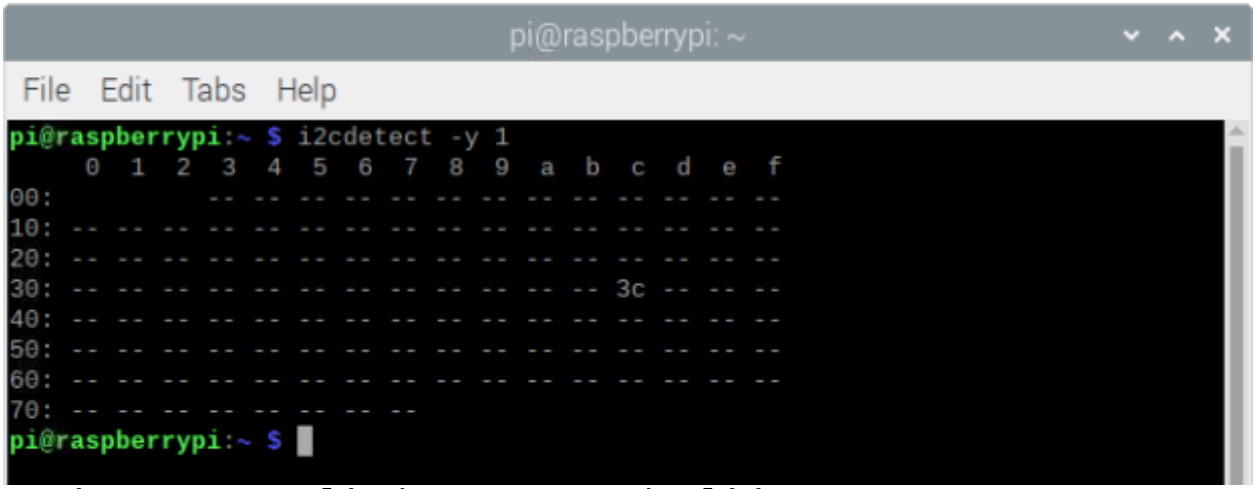
Le répertoire doit être changé en *Adafruit_Python_SSD1306*, en exécutant la commande suivante : **cd Adafruit_Python_SSD1306**

Ensuite, la bibliothèque sera installée en exécutant la commande suivante :

sudo python3 setup.py install

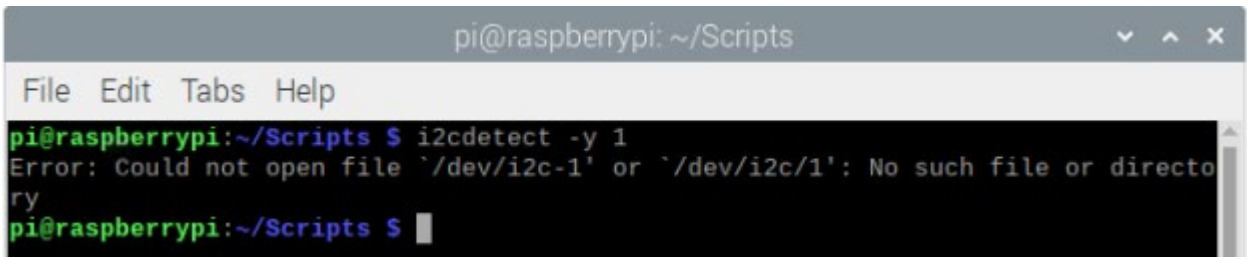
Avant d'utiliser un dispositif connecté à l'interface I2C, l'adresse I2C doit d'abord être détectée. Pour détecter l'adresse I2C de l'écran, la commande suivante doit être exécutée dans le terminal : **i2cdetect -y 1**

Le résultat devrait ressembler à l'image suivante :



Où 0x3c est l'adresse I2C de l'écran.

Si l'interface I2C n'est pas activée avec la commande précédente, l'erreur apparaîtra comme sur l'image suivante :



Il y a plusieurs exemples de scripts fournis avec la bibliothèque, naviguez dans le répertoire : */Adafruit_Python_SSD1306/examples*
en exécutant la commande suivante :
cd ~/*Adafruit_Python_SSD1306*/examples

Ce répertoire contient plusieurs exemples de scripts, notamment :
shapes.py,
image.py,
stats.py
et autres.

L'accent est mis sur le script *shapes.py*. Pour exécuter le script, ouvrez le terminal dans le répertoire où le script est enregistré et exécutez la commande suivante :
python3 shapes.py

Python script

```
import time
import Adafruit_SSD1306
from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont

disp = Adafruit_SSD1306.SSD1306_128_64(rst=None)
disp.begin()
disp.clear()
disp.display()
image = Image.new('1', (disp.width, disp.height))
draw = ImageDraw.Draw(image)

print('[Press CTRL + C to end the script!]')
try:
      while True:
            draw.rectangle((0, 0, disp.width, disp.height),
                  outline=0, fill=0)

            padding = 2
            shape_width = 20
            top = padding
            bottom = disp.height - padding

            print('Drawing a ellipse')
            x = padding
            draw.ellipse((x, top , x + shape_width, bottom),
                  outline=255, fill=0)
            time.sleep(0.2)
```

```
# two tabs
            print('Drawing a rectangle')
            x += shape_width + padding
            draw.rectangle((x, top, x + shape_width, bottom),
                  outline=255, fill=0)
            time.sleep(0.2)

            print('Drawing a triangle')
            x += shape_width + padding
            draw.polygon([(x, bottom), (x + shape_width / 2, top),
                  (x + shape_width, bottom)], outline=255, fill=0)
            time.sleep(0.2)

            print('Drawing two lines')
            x += shape_width + padding
            draw.line((x, bottom, x + shape_width, top), fill=255)
            draw.line((x, top, x + shape_width, bottom), fill=255)
            time.sleep(0.2)

            print('Printing text')
            x += shape_width + padding
            my_font = ImageFont.load_default() # Load default font.
            draw.text((x, top), 'AZ', font=my_font,
                  fill=255)
            draw.text((x, top + 20), 'DLVRY', font=my_font,
                  fill=255)
            time.sleep(0.2)

            disp.image(image)
            disp.display()
            time.sleep(1)
```

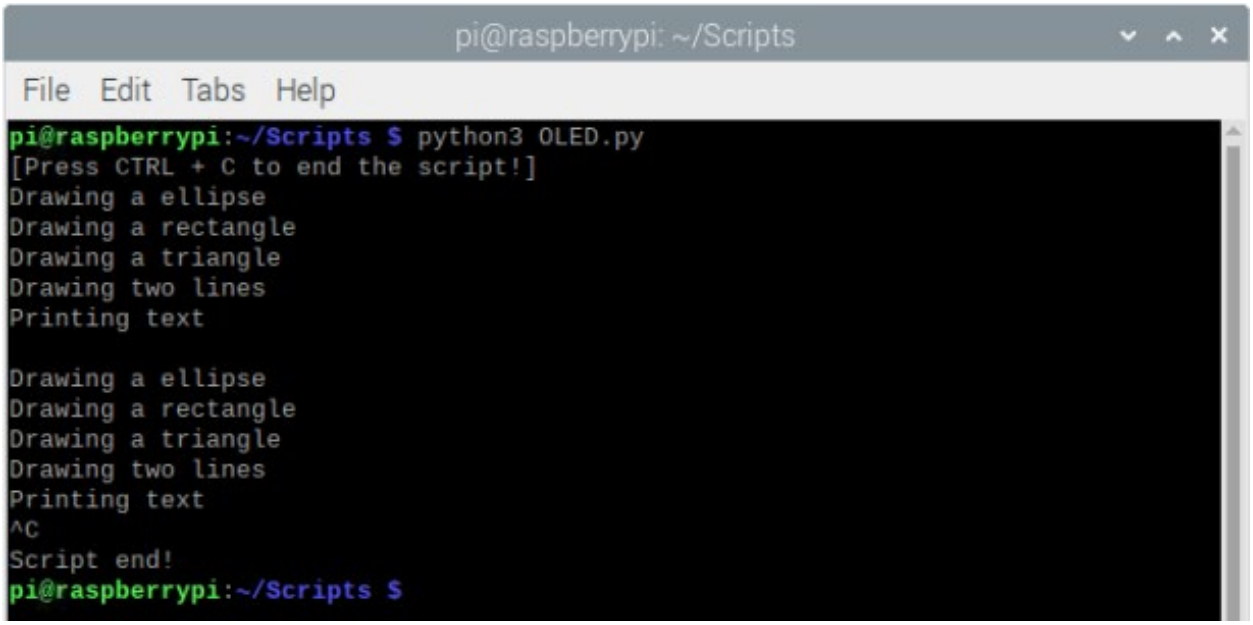
```
# two tabs
            print()
            disp.clear()
            disp.display()

except KeyboardInterrupt:
      print('\nScript end!')

finally:
      disp.clear()
      disp.display()
```

Enregistrez le script sous le nom de "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 plusieurs bibliothèques et fonctions.

Ensuite, un objet appelé disp est créé avec la ligne de code suivante :
`disp = Adafruit_SSD1306.SSD1306_128_64(rst=None)`

Où rst=None est utilisé. Cela représente la broche de réinitialisation, que l'écran OLED de 0,91 pouces ne possède pas. La bibliothèque Adafruit_SSD1306 peut être utilisée pour de nombreux autres écrans OLED, c'est pourquoi il y a une option pour cette broche.

L'objet disp représente l'écran lui-même et cet objet est utilisé pour envoyer des commandes à l'écran.

Ensuite, l'objet écran est initialisé, le mémoire tampon de données de l'écran a été effacé. Après cela, les images sont créées.

Pour créer une image, il faut d'abord créer une image vide aux dimensions de l'écran. Cela se fait avec la ligne de code suivante :
`image = Image.new('1', (disp.width, disp.height))`

Ensuite, avec cet objet image, on crée l'objet draw, qui est utilisé pour dessiner des formes : *`draw = ImageDraw.Draw(image)`*

`Ensuite, le bloc de code try-except-finally est créé. Dans le bloc de code try, la boucle indéfinie est créée (while True :). Dans le bloc de code indéfini se trouve l'algorithme de contrôle de l'écran.`

Le bloc de code *except* qui s'exécute lorsque on appuie sur CTRL + C sur le clavier. C'est ce qu'on appelle une interruption de clavier et elle est utilisée pour mettre fin au script. Lorsque ce bloc de code est exécuté, le message Script end! s'affiche dans le terminal.

Le bloc de code *finally* est exécuté à la fin de l'exécution du script. Il est utilisé pour vider le tampon de données de l'écran et pour désactiver tous les modes et/ou interfaces de broches GPIO utilisés.

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, dont les deux premiers sont les positions X et Y du coin supérieur droit du rectangle. Le troisième élément est la largeur du rectangle et le quatrième est la hauteur 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, le noir - le pixel est éteint (OFF), et le 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 255, 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 droit du rectangle qui contient l'ellipse. Le troisième élément est la largeur du rectangle et le quatrième élément est la hauteur du rectangle. Si la largeur est égale à la hauteur, la forme qui est dessinée est le cercle. Le deuxième

argument est l'argument de contour, et le troisième argument est l'argument de contenu.

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 et représentent un point d'angle. Les éléments du tuples représentent la position *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 dessinée. 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 la position *X* et *Y* du point de départ d'une ligne et les deux seconds éléments représentent la position *X* et *Y* du point d'arrivée d'une ligne. Le deuxième argument est l'argument de contenu.

Pour afficher le texte, on utilise la fonction *text()*. Cette fonction accepte quatre arguments et ne renvoie aucune valeur. Le premier argument est un tuple de deux

éléments, qui représentent les positions *X* et *Y* du curseur où le texte est affiché. Le deuxième argument représente le texte lui-même, (une valeur de chaîne). Le troisième argument est l'argument police, et le quatrième argument est l'argument remplissage. L'argument police représente la police utilisée. Pour définir la valeur de l'argument police, la ligne de code suivante est utilisée avec une police de bibliothèque par défaut : font = ImageFont.load_default()
Il est possible d'utiliser d'autres polices, mais cela n'est pas abordé dans cet eBook.

Il est temps d'apprendre et de réaliser des projets par vous-même. 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 pour

Raspberry Pi, 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