¡Bienvenido!

Muchas gracias por comprar nuestra *AZ*-Delivery Pantalla OLED I2C de 1.3 pulgadas. En las siguientes páginas se presentará como utilizar y configurar este práctico dispositivo.

¡Diviértase!

Tabla de Contenidos

Introducción

OLED son las siglas de un diodo orgánico de emisión de luz. Las pantallas OLED son un conjunto de LEDs unidos en una matriz. La pantalla OLED de 1.3 pulgadas tiene 128x64 píxeles (LEDs). Para controlar estos LEDs se necesita un circuito controlador o un chip. La pantalla tiene un chip controlador *SH1106*. El chip controlador tiene una interfaz I2C que permite la comunicación con el microcontrolador principal.

La pantalla OLED y el chip controlador SH1106 operan en el rango de *3.3V*. En el módulo se encuentra un regulador de voltaje de *3.3V*, esto permite que la pantalla pueda funcionar en el rango de *5V*.

El rendimiento de estas pantallas es mucho mayor que el de las LCD tradicionales. La simple comunicación I2C y el bajo consumo de energía, permite que sean más adecuadas para una variedad de aplicaciones.

Especificaciones

Voltaje de la fuente de alimentación	de 3.3V a 5V
Interfaz de comunicación	I2C
Color de pixel	Blanco
Temperatura de funcionamiento	de -20 a 70 ℃
Bajo consumo de energía	< 11mA
Dimensiones	36x34x3mm [1.4x1.3x 0.1pulgadas]

Para prolongar la vida útil de la pantalla, es común utilizar un “Screen saver”. Se recomienda no utilizar información constante durante un largo período de tiempo, porque esto acortará la vida útil de la pantalla y aumentará el efecto de “Screen burn”.

Configuración dle Arduino IDE

Si el Arduino IDE no está instalado, seguir el siguiente *link* y descargar el archivo de instalación para el sistema operativo de su elección.

Para los usuarios de *Windows*, hacer doble clic en el archivo descargado *.exe* y seguir las instrucción de la ventana de instalación.

Para los usuarios de *Linux*, descargar un archivo con la extensión *.tar.xz*, que se debe extraer. Cuando se la extrae, ir al directorio extraído y abrir el terminal en ese directorio. Se deben ejecutar dos scripts *.sh,* el primero es *arduino-linux-setup.sh* y el segundo es *install.sh*.

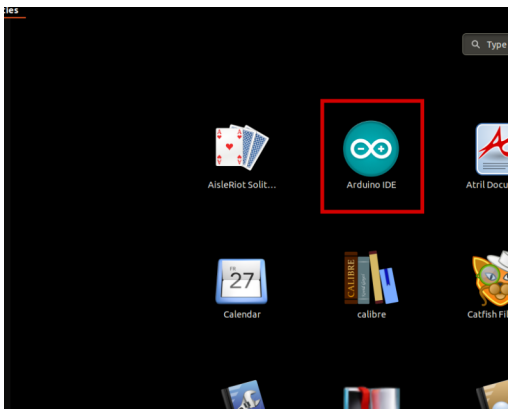
Para ejecutar el primer script en el terminal, abrir el terminal en el directorio extraído y ejecutar el siguiente comando:

sh arduino-linux-setup.sh user_name

user_name - es el nombre de un superusuario en el sistema operativo Linux. Una contraseña para el superusuario se debe introducir cuando se inicia el comando. Se debe esperar unos minutos para que el script complete todo.

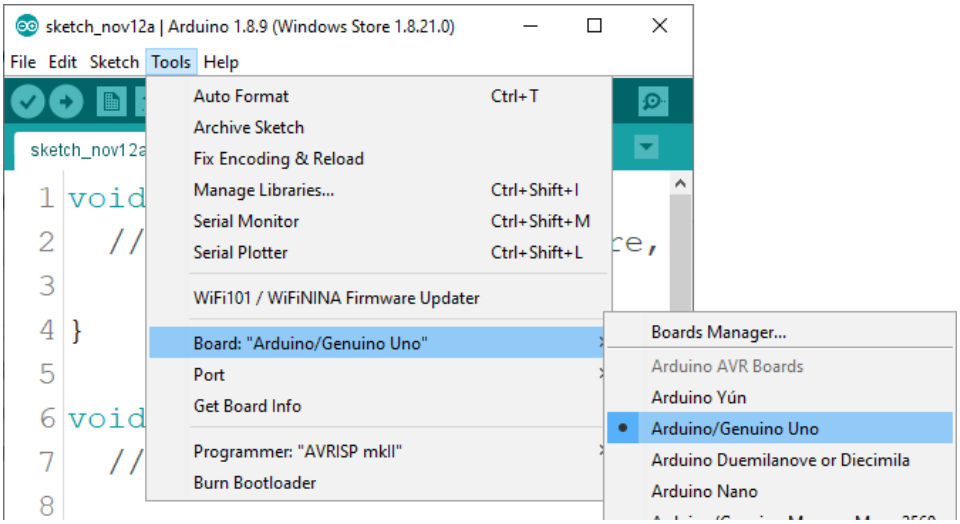
El segundo script *install.sh* se debe utilizar después de la instalación del primer script. Ejecutar el siguiente comando en el terminal (directorio extraído): **sh install.sh**

Después de la instalación de estos scripts, ir a la sección *All Apps*, donde *Arduino IDE* está instalado.



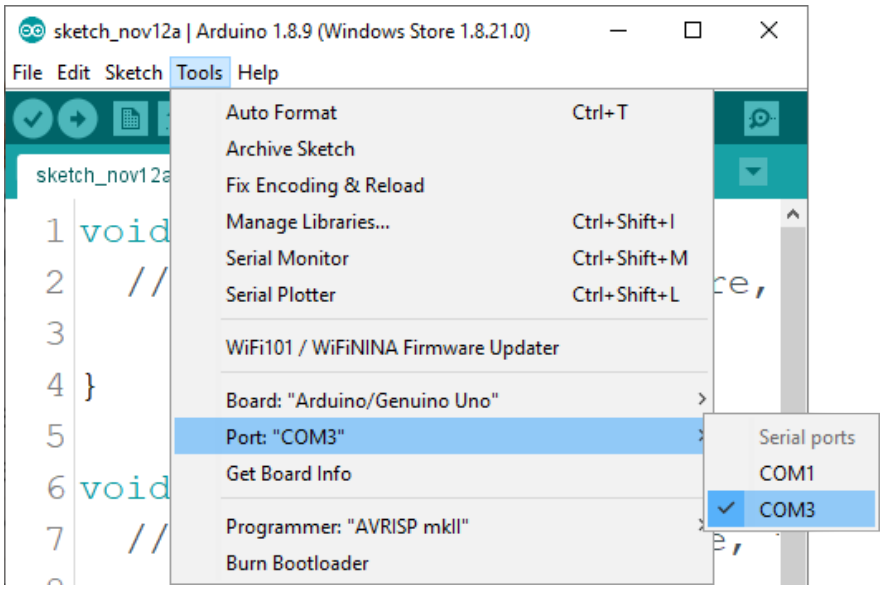
La mayoría de los sistemas operativos incluyen un editor de texto preinstalado (por ejemplo, *Windows* incluye *Notepad*, *Linux Ubuntu* incluye *Gedit*, *Linux Raspbian* incluye *Leafpad*, etc.). Todos estos editores de texto se pueden utilizar perfectamente para el objetivo de este E-book.

Lo siguiente es comprobar si el PC puede detectar una placa Atmega328P Board. Para esto, se debe abrir el Arduino IDE recién instalado, e ir a:
Tools > Board > {your board name here}
{your board name here} debe ser el *Arduino/Genuino Uno*, como se puede observar en la imagen a continuación:



Se debe seleccionar el puerto al que está conectado la placa de Atmega328P Board. Ir a: *Tools > Port > {port name goes here}*
y cuando la placa de Atmega328P Board está conectada al puerto USB, el nombre del puerto se puede observar en el menú desplegable de la imagen anterior.

Si el Arduino IDE se utiliza en *Windows*, los nombres de los puertos son los siguientes:



Para los usuarios de *Linux,* por ejemplo, el nombre del puerto es */dev/ttyUSBx*, donde *x* representa un número entero entre *0* y *9*.

Configuración de Raspberry Pi y Python

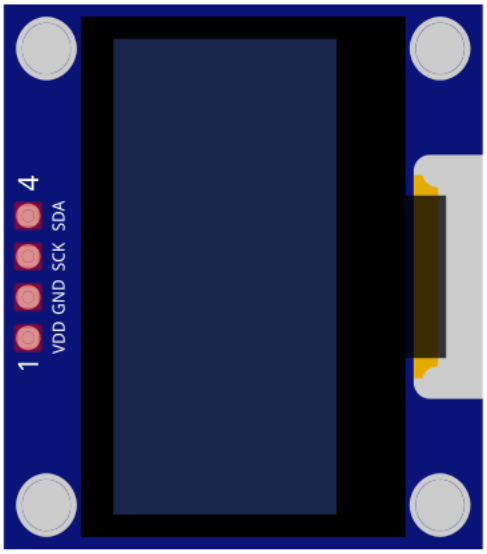
Para la configuración de la Raspberry Pi, primero se debe instalar el sistema operativo, después se debe configurar todo para que se pueda utilizar en el modo *Headless*. El modo *Headless* permite la conexión remota a la Raspberry Pi, sin la necesidad de un monitor de pantalla de PC, ratón o teclado. Las únicas opciones que se utilizan en este modo son la propia Raspberry Pi, la fuente de alimentación y la conexión a internet. Todo esto se explica muy detalladamente en el E-book gratuito:
Raspberry Pi Quick Startup Guide

El sistema operativo *Raspbian* incluye *Python* preinstalado.

La disposición de los pines

La pantalla OLED de 1.3 pulgadas tiene cuatro pines. La disposición de los pines se observa en la siguiente imagen:

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

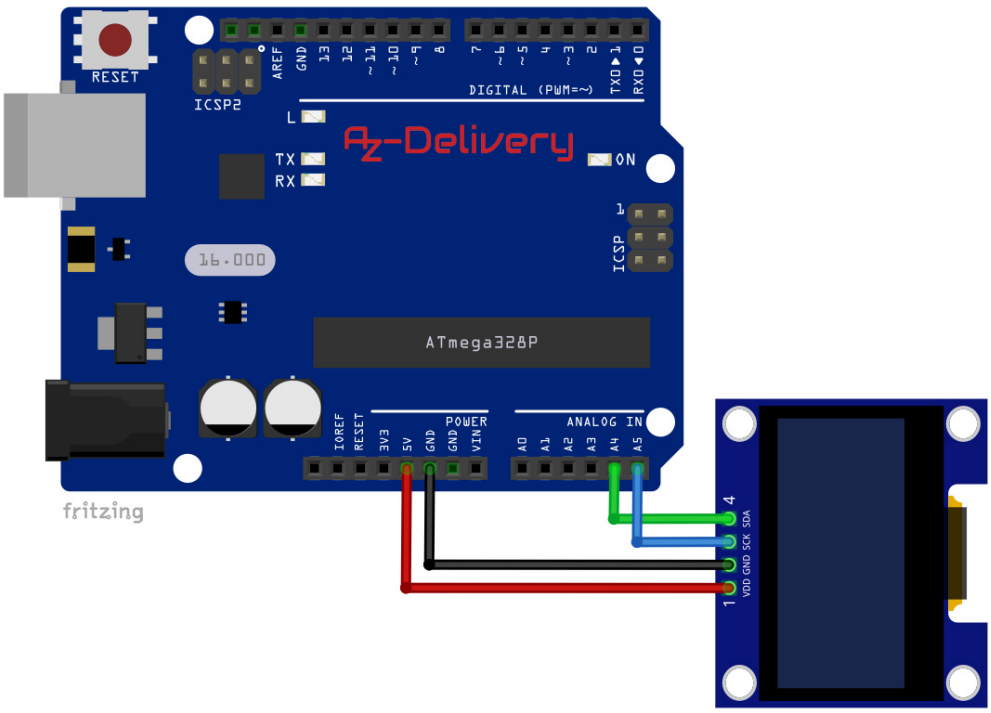


La pantalla incluye un regulador de voltaje. Los pines de la pantalla OLED de *1.3* pulgadas se pueden conectar a una fuente de alimentación de *3.3V o 5V* sin peligro para el propio sensor.

NOTA: Cuando se utiliza la Raspberry Pi, la fuente de alimentación se debe extraer solamente de un pin 3.3V.

Conectando la pantalla con Atmega328P Board

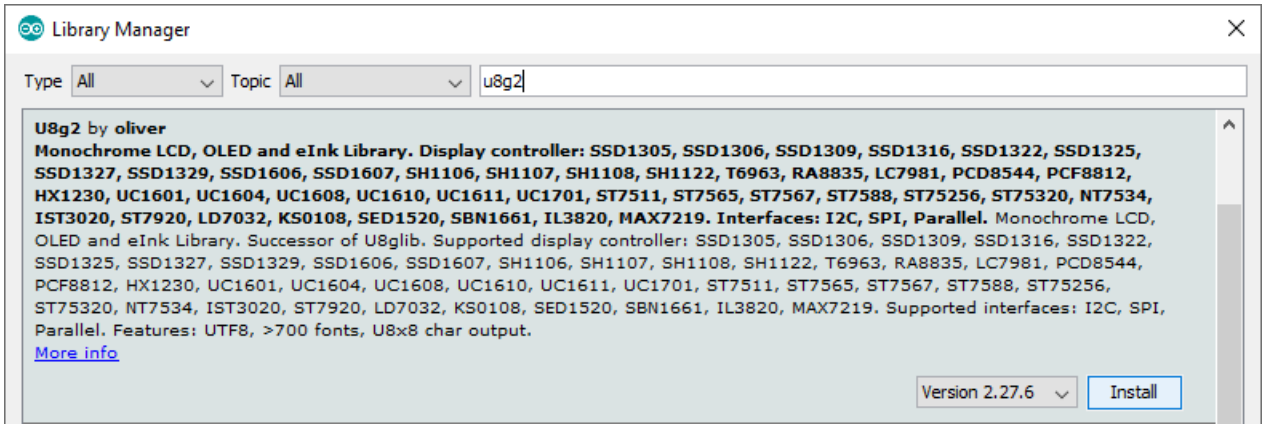
Conectar la pantalla OLED de 1.3 pulgadas con el Atmega328P Board como se observa en el siguiente diagrama de conexión:



Pin de Pantalla	Pin	Color de cable
SDA	A4	**Cable verde**
SCK	A5	**Cable azul**
GND	GND	**Cable negro**
VCC	5V	**Cable rojo**

Librería para Arduino IDE

Para utilizar la pantalla con Atmega328P Board, se recomienza descargar una librería externa para esto. La librería que se va a utilizar es “*U8g2”*. Para descargarla e instalarla, abrir Arduino IDE e ir a: *Tools* > *Manage Libraries*. Cuando se abra una nueva ventana, escribir “*u8g2”* en el cuadro de búsqueda e instalar la librería “*U8g2”* diseñada por “*oliver”*, como se observa en la siguiente imagen:



Muchos ejemplos de sketchs se incluyen con la librería, para abrir uno, ir a:
File > *Examples* > *U8g2* > *full_buffer* > *GraphicsTest*
Con este ejemplo de sketch, se puede probar la pantalla. En este E-Book, el código del sketch se modifica para crear una versión del código más amigable.

Ejemplo del sketch

```
#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);
}
```

Primero, se importan dos librerías, la *U8g2lib* y *Wire*. A continuación, se crea un objeto denominado *u8g2* con la siguiente línea de código:

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

El objeto creado representa la propia pantalla y se utiliza para controlar la pantalla. La librería *U8g2* se puede utilizar para otras pantallas OLED, por lo que se encuentran muchos constructores en los ejemplos de sketchs de la librería.

A continuación, se crea la función denominada *u8g2_prepare()* que no tiene argumentos y no devuelve ningún valor. Se utilizan las cinco funciones de la librería *u8g2*.

La primera función se denomina *setFont()*, que tiene un argumento y no devuelve ningún valor. El argumento representa la fuente *u8g2*. La lista de fuentes disponibles se encuentra en el siguiente enlace:
https://github.com/olikraus/u8g2/wiki/fntlist8x8

La segunda función se denomina *setFontRefHeightExtendedText()*, que no tiene argumentos y no devuelve ningún valor. Se utiliza para dibujar caracteres en la pantalla. Una explicación más detallada, se puede encontrar en el siguiente enlace:
https://github.com/olikraus/u8g2/wiki/u8g2reference#setfontrefheightextendedtext

La tercera función se denomina *`setDrawColor()`*, que tiene un argumento y no devuelve ningún valor. El valor del argumento es un número entero que representa un índice de color para todas las funciones de dibujo. Los procedimientos de dibujo de fuentes utilizan este argumento para establecer el color del primer plano. El valor predeterminado es *1.* Si se establece en *0*, el espacio alrededor del carácter se ilumina, y el carácter no. También se puede utilizar el valor del argumento *2*, aunque no se diferencia con el *0*.

La cuarta función se denomina *`setFontPosTop()`*, que no tiene ningún argumento y no devuelve ningún valor. Esta función controla la posición del carácter en una línea de texto. La función tiene tres versiones. La primera es *`setFontPosBaseLine(),`* la segunda es *`setFontPosCenter()`*, la tercera es *`setFontPosBottom().`* El objetivo es cambiar la posición de los caracteres en una línea.

La quinta función se denomina *`setFontDirection()`*, que tiene un argumento y no devuelve ningún valor. El argumento es un número entero que representa la dirección del texto. El valor es un número entero en el rango de *0* a *3* (*0 = 0°*, *1 = 90°*, *2 = 180°* y *3 = 270°*).

La función *drawStr()* tiene tres argumentos y no devuelve ningún valor. Se utiliza para mostrar una cadena constante en la pantalla. Los dos primeros argumentos representan las posiciones *X* e *Y* del cursor, donde se muestra el texto. El tercer argumento representa el propio texto, un valor de cadena. Las funciones que configuran la disposición del texto se deben usar antes de utilizar la función *drawStr()*, o de lo contrario la función *drawStr()* utilizará los ajustes predeterminados para la fuente, el tamaño y la disposición general del texto.

Para mostrar las formas, se deben utilizar funciones de la librería específicas para cada forma:

La función *drawFrame()* tiene cuatro argumentos y no devuelve ningún valor. Se utiliza para mostrar un marco, un rectángulo vacío. Los dos primeros argumentos representan las posiciones *X* e *Y* de la esquina superior izquierda del marco. El tercer argumento representa el ancho del marco y el cuarto argumento representa el alto del marco.

La función *drawRFrame()* tiene cinco argumentos y no devuelve ningún valor. Se utiliza para mostrar un marco con esquinas redondas. Los dos primeros argumentos representan las posiciones *X* e *Y* de la esquina superior izquierda del marco. Los siguientes dos argumentos representan el ancho y el alto del marco, y el quinto argumento representa el radio de la esquina.

La función *drawBox()* tiene cuatro argumentos y no devuelve ningún valor. Se utiliza para mostrar un rectángulo relleno. Los dos primeros argumentos representan las posiciones *X* e *Y* de la esquina superior izquierda del rectángulo. Los siguientes dos argumentos representan el ancho y el alto del rectángulo.

La función *drawRBox()* tiene cinco argumentos y no devuelve ningún valor. Se utiliza para mostrar un rectángulo relleno con bordes redondos. Los dos primeros argumentos representan las posiciones *X* e *Y* de la esquina superior izquierda del rectángulo. Los siguientes dos argumentos representan el ancho y el alto del rectángulo y el quinto argumento representa el radio de la esquina.

La función *drawCircle()* tiene tres argumentos y no devuelve ningún valor. Se utiliza para mostrar un círculo. Los dos primeros argumentos representan las posiciones *X* e *Y* del punto central del círculo. El tercer argumento representa el radio del círculo.

La función *drawDisc()* tiene tres argumentos y no devuelve ningún valor. Se utiliza para mostrar un disco. Los dos primeros argumentos representan las posiciones *X* e *Y* del punto central del disco. El tercer argumento representa el radio del disco.

La función *drawTriangle()* tiene seis argumentos y no devuelve ningún valor. Se utiliza para mostrar un triángulo relleno. Los dos primeros argumentos representan las posiciones *X* e *Y* del primer punto de la esquina del triángulo. Los siguientes dos argumentos representan las posiciones *X* e *Y* del segundo punto de la esquina del triángulo. Los dos últimos argumentos representan las posiciones *X* e *Y* del último punto de la esquina del triángulo.

La función *drawLine()* tiene cuatro argumentos y no devuelve ningún valor. Se utiliza para mostrar una línea. Los dos primeros argumentos representan las posiciones *X* e *Y* del punto inicial de la línea. Los siguientes dos argumentos representan las posiciones *X* e *Y* del punto final de la línea.

La función *drawUTF8()* tiene tres argumentos y devuelve un valor. Se utiliza para mostrar un texto, el valor de la cadena que puede contener un carácter codificado como un carácter *Unicode*. Los dos primeros argumentos representan las posiciones X e Y del cursor y el tercero representa el propio texto. Los caracteres *Unicode* se pueden mostrar de varias maneras. La primera es copiando y pegando el carácter existente en el sketch, como en la siguiente línea del código:

```
u8g2.drawUTF8(50, 20, "☂");
```

La segunda es crear una matriz *char*, que tiene dos valores: el primer valor es un número hexadecimal del carácter *Unicode* y el segundo valor es

un carácter nulo (“*\0*”). Esto se puede realizar usando la matriz *char* denominada *COPYRIGHT_SYMBOL*, como en las siguientes líneas del código:

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

La tercera forma de usar la función es utilizar un número hexadecimal para el propio carácter, como en la siguiente línea de código:

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

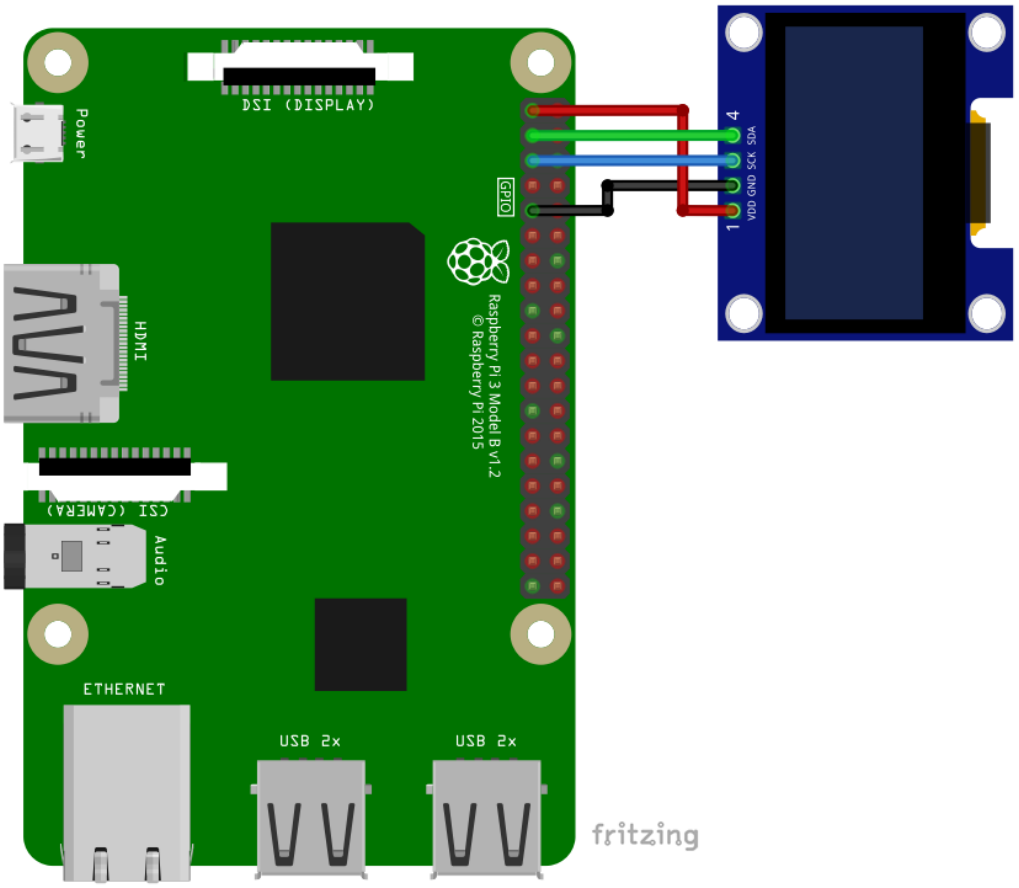
La función devuelve un valor, un número entero que representa el ancho del texto (*cadena*).

Para mostrar algo en la pantalla, se tiene que borrar el buffer de datos de la pantalla, luego se debe establecer un nuevo valor para el buffer de datos (una imagen) y enviar el nuevo valor a la pantalla. De esta manera, una nueva imagen se mostrará en la pantalla. Para que este cambio sea visible, la función *delay()* se tiene que utilizar para retrasar el próximo cambio del buffer de datos, como en las siguientes líneas de código:

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

Conectando la pantalla con Raspberry Pi

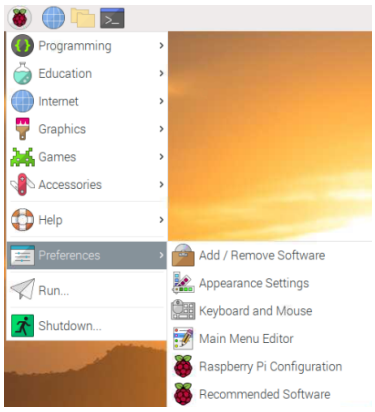
Conectar la pantalla con la Raspberry Pi como se observa en el siguiente diagrama de conexión:



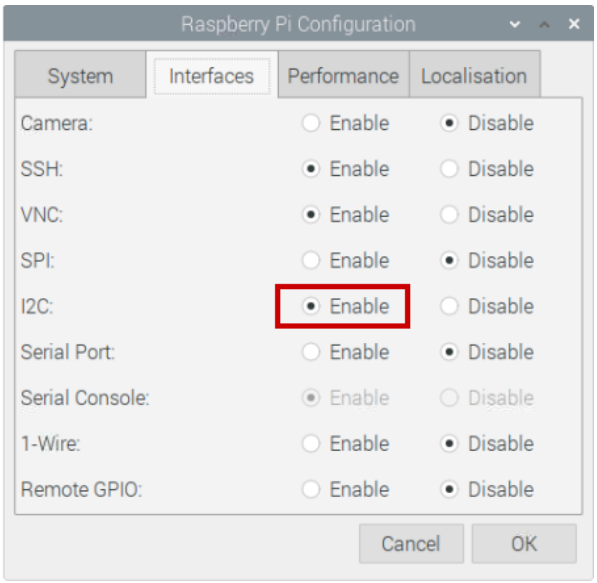
Pin de Pantalla	Pin de Raspberry Pi	Pin Físico	Color de cable
SDA	GPIO2	3	**Cable verde**
SCL	GPIO3	5	**Cable azul**
GND	GND	9	**Cable negro**
VCC	3V3	1	**Cable rojo**

Habilitando la interfaz I2C

Para poder utilizar la pantalla con Raspberry Pi, la interfaz I2C debe estar habilitada. Abrir el siguiente menú:
Application Menu > Preferences > Raspberry Pi Configuration



En la nueva ventana, bajo la pestaña *Interfaces*, habilitar el botón de radio I2C, como se observa en la siguiente imagen:



Librerías y herramientas para Python

Para utilizar la pantalla OLED de 1.3 pulgadas con una Raspberry Pi se recomienda descargar una librería externa. La librería que se va a utilizar se denomina "*luma.oled*", pero antes de descargarla, se deben instalar herramientas para Python.

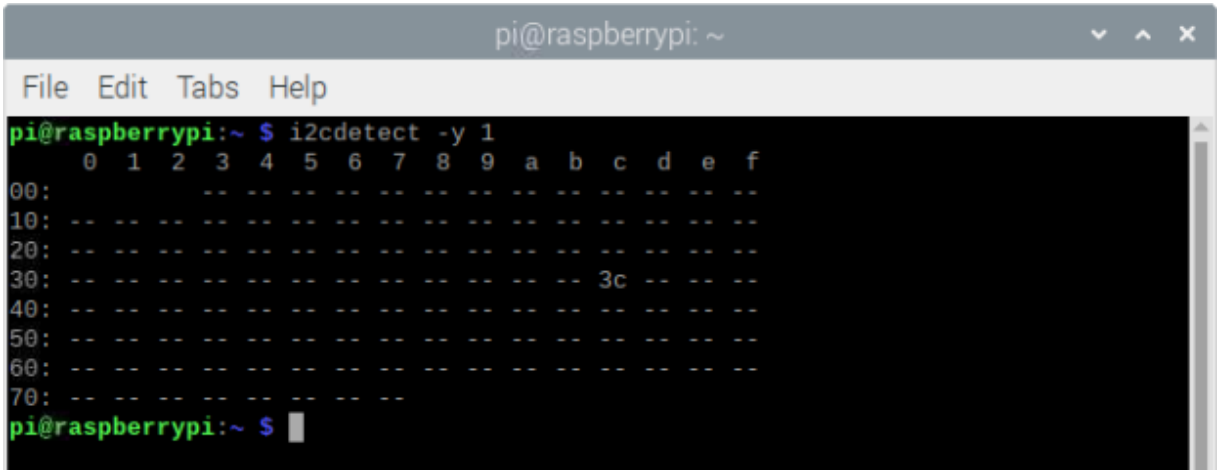
Abrir el terminal y ejecutar el siguiente comando:

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

A continuación, la pantalla de la dirección I2C se debe detectar. En la ventana del terminal, ejecutar el siguiente comando:

```
i2c detect -y 1
```

El resultado se debe observar como la salida de la siguiente imagen:



donde *0x3c* es la dirección I2C de la pantalla.

Para instalar la librería *luma.oled,* abrir el terminal y ejecutar el siguiente comando:

sudo -H pip3 install –upgrade luma.oled

El último paso es descargar los scripts de ejemplo de *GitHub*. En la ventana del terminal, ejecutar el siguiente comando:

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

Luego, (en el terminal) cambiar el directorio al directorio *luma.examples*, con el siguiente comando: **cd luma.examples**

y para instalarlo, ejecutar el siguiente comando:

sudo -H pip3 install -e .

(con el punto al final)

Python script

El siguiente código script es un código modificado de dos scripts: */luma.examples/demo.py* y */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)
```

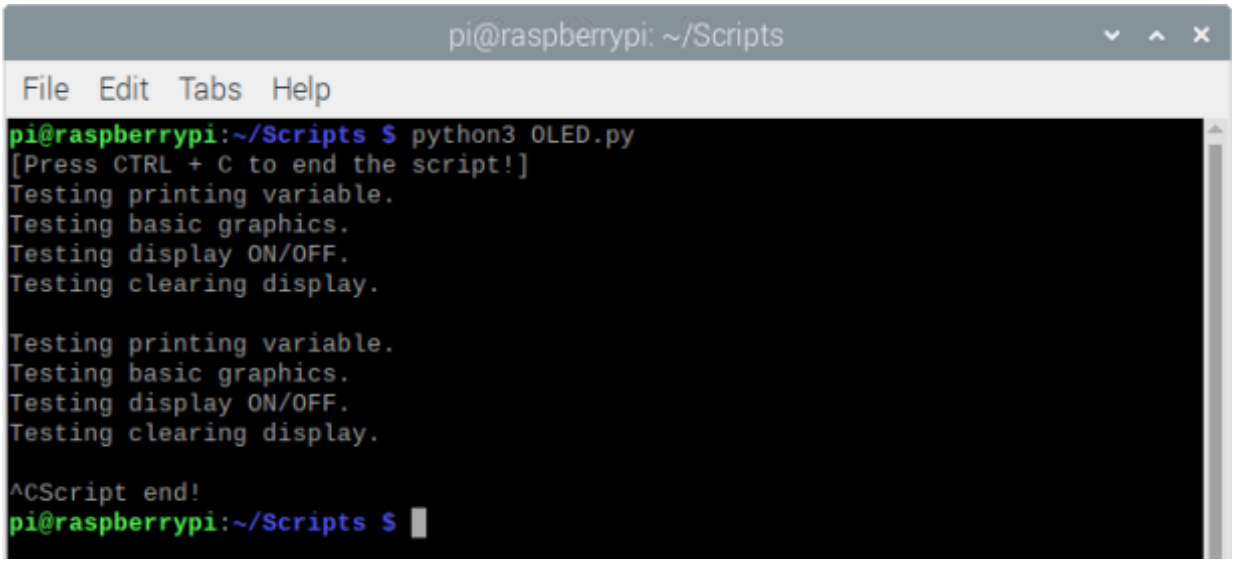
```
def primitives(device):
    with canvas(device) as draw:
        # Draw a rectangle.
        draw.rectangle((4, 4, 40, 10), outline=1, fill=0)

        # 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!')
```

Guardar el script con el nombre "*OLED.py*". Para ejecutar el script, abrir el terminal en el directorio donde el script está guardado y ejecutar el siguiente comando:

python3 OLED.py

El resultado se debe observar como la salida de la siguiente imagen:



Para detener el script, presionar "CTRL + C" en el teclado.

El script comienza con la importación de librerías y funciones.

Después el camino está guardado en la fuente especial de la variable *font_path*. La fuente que se utiliza es *ChiKareGo.ttf*, que se puede encontrar en: *luma.examples > examples > fonts > ChiKareGo.ttf*
El archivo de la fuente se debe copiar en el mismo directorio donde el script está guardado.

Después, se crean dos funciones, la primera es *changing_var()* y la segunda es *primitives()*.

La función *changing_var()* se utiliza para mostrar una variable que cambia de estado cada milisegundo. La función acepta un argumento y no devuelve ningún valor. El argumento es un argumento *device*, que se explicará más adelante en el texto. Al principio de la función *changing_var()*, la variable *size* se define y se inicializa con el número *40*. Este valor representa el tamaño de la fuente. Después, se crea una variable *nf* (*new_font*), que representa la fuente que se utiliza. Las variables *size* y *font_path* se utilizan para crear la variable *nf*. Después, el loop *for* se crea para hacer un lazo a través de los primeros cien números enteros. Estos valores se utilizan como valores para la variable que se mostrará en la pantalla. En el loop *for*, se crea una imagen que se mostrará en la pantalla, con la siguiente línea del código:
`with canvas(device) as draw:`
un objeto *draw* se crea. El objeto *draw* representa su propia imagen.

Para emitir el texto, se utiliza la función *text()*. La función *text()* acepta cuatro argumentos, de los cuales uno es opcional, y no devuelve ningún valor. El primer argumento es un *tuple* con dos elementos, los elementos representan las posiciones *X* e *Y* del cursor. El segundo argumento representa el propio texto, un valor de cadena. El tercer argumento es el argumento *fill*. El cuarto argumento es opcional, es el argumento *font*. Si no se utiliza el argumento, la fuente se establecerá con la fuente y el tamaño predeterminados. Sin embargo, si la variable *nf* se almacena en el argumento *font*, se puede utilizar la fuente preferida y cambiar el tamaño de la fuente.

El argumento *fill* representa el color del texto. Si *fill = 0* el color será negro (no se muestra nada en la pantalla) y si *fill* es igual a cualquier otro valor, por ejemplo *fill = 1*, el color del texto será blanco (las pantallas OLED sólo tienen colores blanco y negro). Esta librería también se puede utilizar para otras pantallas, por lo que en el argumento *fill*, se puede almacenar cualquier otro color, lo que no es posible para una pantalla OLED de 1.3 pulgadas.

El resto del código dentro del loop *for* es un algoritmo para mostrar el texto y la variable con número de dos dígitos desde *00* al *99*.

La segunda función es *primitives()* y se utiliza para dibujar formas en la pantalla. Esta función acepta un argumento, el argumento *device*, no devuelve ningún valor. Al principio de la función, se crea la imagen del

objeto *draw*. Este objeto se utiliza para dibujar muchas formas.

Para dibujar el rectángulo, se utiliza la función *rectangle().* La función acepta tres argumentos y no devuelve ningún valor. El primer argumento es un *tuple* de cuatro elementos, donde los dos primeros elementos representan las posiciones *X* e *Y* de la esquina superior derecha del rectángulo. El tercer y cuarto elemento representan las posiciones *X* e *Y* de la esquina inferior derecha del rectángulo. El segundo argumento es el argumento *outline* y el tercer argumento es el argumento *fill*.

El valor de la posición *X* comienza en el lado izquierdo de la pantalla (valor de *0*) y termina en el lado derecho de la pantalla (valor de *127*). El valor de la posición *Y* comienza en la parte superior de la pantalla (valor de *0*) y termina en la parte inferior de la pantalla (valor de *63*).

El argumento *outline* representa el color del borde de la forma y el argumento *fill* representa el color de la propia forma. Debido a que se utilizan las pantallas OLED, los colores son blanco y negro, negro – el píxel está apagado, blanco – el píxel está encendido. Cuando se guarda el valor cero en el argumento *outline* o en el argumento *fill* significa que es un color negro. Cuando se guarda cualquier otro valor mayor que cero, por ejemplo *1*, representa el color blanco.

Para dibujar elipses o círculos se utiliza la función *ellipse()*. La función acepta tres argumentos y no devuelve ningún valor. El primer argumento es un *tuple* de cuatro elementos. Los primeros dos elementos representan las posiciones *X* e *Y* de la esquina superior izquierda del rectángulo que contiene la elipse. Los siguientes dos elementos representan las posiciones *X* e *Y* de la esquina inferior derecha del rectángulo que contiene la elipse. El segundo argumento es el argumento *outline* y el tercer argumento es el argumento *fill*.

Para dibujar un polígono, se utiliza la función *polygon()*. Un polígono es un triángulo, un rectángulo o cualquier otra forma con 3 o más esquinas. La función acepta tres argumentos. El primer argumento es una *list* de tres o más *tuples*. Los *tuples* tienen dos elementos, que representan las posiciones X e Y del punto de esquina de la forma. El número de *tuples* en *list* es arbitrario, tres o más *tuples*, lo que depende de la forma que se quiera dibujar. El segundo argumento es el argumento *outline* y el tercer argumento es el argumento *fill*.

Para dibujar una línea, se utiliza la función *line()*. La función acepta dos argumentos y no devuelve ningún valor. El primer argumento es un *tuple* de cuatro elementos, donde los dos primeros elementos representan las posiciones X e Y del punto inicial de una línea y los dos siguientes elementos representan las posiciones *X* e *Y* del punto final de una línea. El segundo argumento es el argumento *fill*.

Al final de la función *primitives(),* la función *text()* se utiliza con y sin argumento *font* para mostrar la diferencia.

Después de estas dos funciones, se crea un bloque de código *try-except.* En el bloque de código *try*, se crean dos objetos y el loop indefinido.

El primer objeto se denomina *serial* y se utiliza para configurar la interfaz I2C y la dirección para la pantalla. Para inicializar este objeto, se utiliza la siguiente línea del código:

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

El segundo objeto se denomina *device*. Este objeto representa el propio chip conductor, y se utiliza para controlar el chip conductor. Para inicializar el objeto del dispositivo *serial*, se utilizan los argumentos *width* y *height*, en la siguiente línea de código:

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

A continuación, se crea un loop indefinido (*while True:*). Dentro del loop indefinido, se ejecutan dos funciones y se prueban dos propiedades de la pantalla. Primero, se ejecuta la función *changing_var(),* luego la función *primitives()*. Después de esto, se utiliza *device.hide()* para apagar la pantalla y *device.show()* para encenderla de nuevo. Al final del loop indefinido, se utiliza *device.clear()* para limpiar el buffer de datos de la pantalla, que también limpia la imagen de la pantalla.

El bloque de código *except* se ejecuta cuando se pulsa 'CTRL+C' en el teclado.
Esto se conoce como la interrupción del teclado. Cuando se ejecuta el bloque de código *except*, se imprime el mensaje "*Script end!*" en el terminal.

Ahora es el momento de aprender y hacer sus propios proyectos. Puede hacerlo con la ayuda de muchos scripts de ejemplo y otros tutoriales, que puede encontrar fácilmente en Internet.

Si está buscando productos de alta calidad para Raspberry Pi, AZ-Delivery Vertriebs GmbH es la compañía adecuada donde podrá obtenerlos. Se le proporcionarán numerosos ejemplos de aplicación, guías completas de instalación, libros electrónicos, librerías y la asistencia de nuestros expertos técnicos.

https://az-delivery.de

¡Disfrute!

Impressum

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