Welcome!

Thank you for purchasing our *AZ-Delivery 1.3 inch OLED I2C Screen*. On the following pages, you will be introduced to how to use and set up this handy device.

Have fun!

Table of Contents

Introduction

OLED stands for Organic Light Emitting Diodes. OLED screens are arrays of LEDs stacked together in a matrix. The 1.3 OLED screen has 128x64 pixels (LEDs). To control these LEDs we need a driver circuit or a chip. The screen has a driver chip called *SH1106*. The driver chip has an I2C interface for communication with the main microcontroller.

The OLED screen and SH1106 driver chip operate in the *3.3V* range. But there is an on-board *3.3V* voltage regulator, therefore, these screens can be operated in the *5V* range.

The performance of these screens is much better than traditional LCDs. Simple I2C communication and low power consumption make them more suited for a variety of applications.

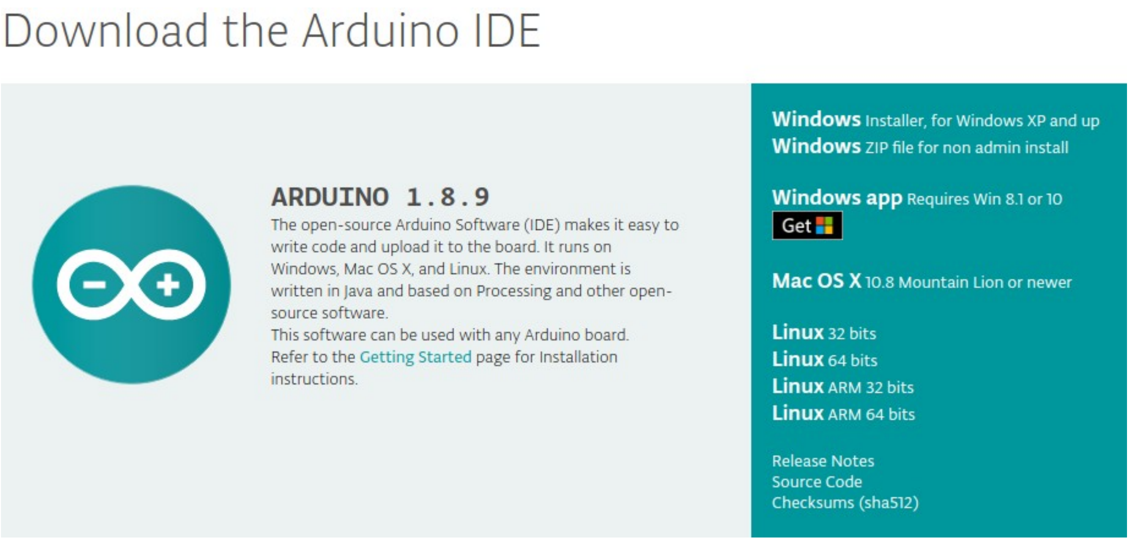
Specifications

Power supply voltage	from 3.3V to 5V
Communication interface	I2C
Pixel Color	White
Operating temperature	from -20 to 70 °C
Low power consumption	< 11mA
Dimensions	36 x 34 x 3mm [1.4 x1.3 x 0.1inch]

To extend the lifetime of the screen, it is common to use a “Screen saver”. It is recommended not to use constant information over a long period of time, because that will shorten the lifespan of the screen and increase the, so called, “Screen burn” effect.

How to set-up Arduino IDE

If the Arduino IDE is not installed, follow the link and download the installation file for the operating system of choice.



For *Windows* users, double click on the downloaded *.exe* file and follow the instructions in the installation window.

For *Linux* users, download a file with the extension *.tar.xz*, which has to be extracted. When it is extracted, go to the extracted directory and open the terminal in that directory. Two *.sh* scripts have to be executed, the first called *arduino-linux-setup.sh* and the second called *install.sh*.

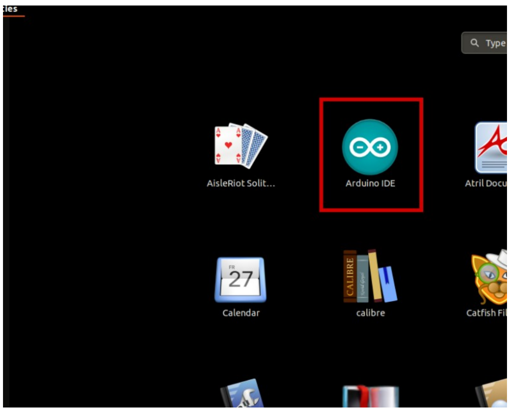
To run the first script in the terminal, open the terminal in the extracted directory and run the following command:

sh arduino-linux-setup.sh user_name

user_name - is the name of a superuser in the Linux operating system. A password for the superuser has to be entered when the command is started. Wait for a few minutes for the script to complete everything.

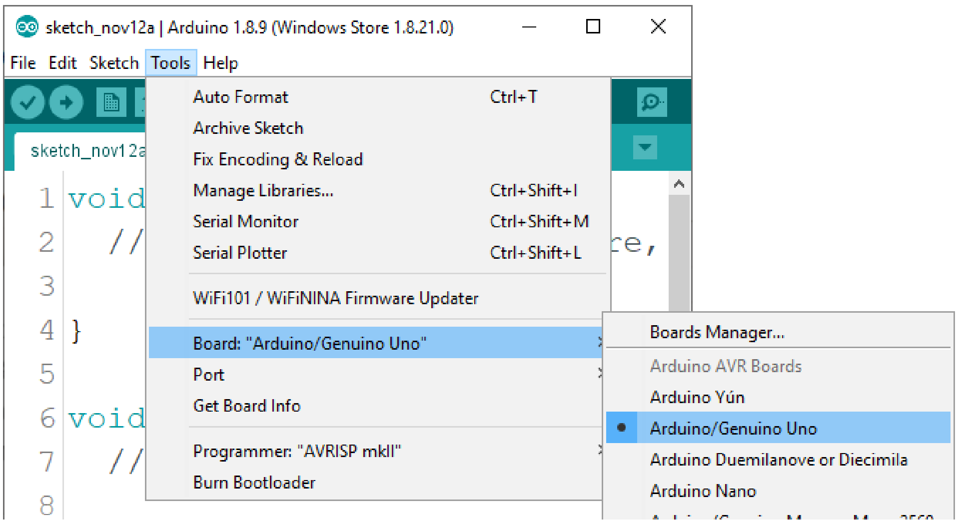
The second script called *install.sh* script has to be used after installation of the first script. Run the following command in the terminal (extracted directory): **sh install.sh**

After the installation of these scripts, go to the *All Apps*, where the *Arduino IDE* is installed.



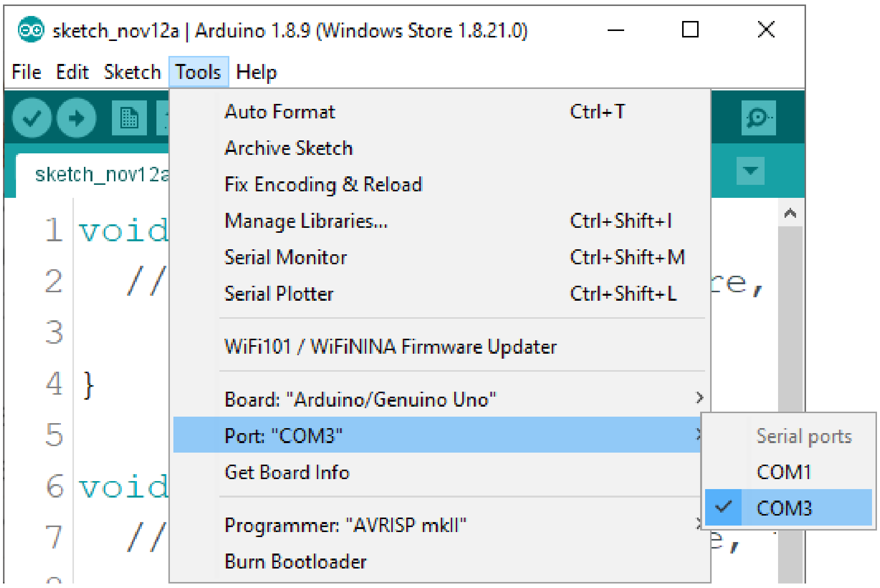
Almost all operating systems come with a text editor preinstalled (for example, `Windows` comes with `Notepad`, `Linux Ubuntu` comes with `Gedit`, `Linux Raspbian` comes with `Leafpad`, etc.). All of these text editors are perfectly fine for the purpose of the eBook.

Next thing is to check if your PC can detect your board. Open freshly installed Arduino IDE, and go to:
Tools > Board > {your board name here}
{your board name here} should be the `Arduino/Genuino Uno`, as it can be seen on the following image:



The port to which the microcontroller board is connected has to be selected.
Go to: *Tools > Port > {port name goes here}*
and when the microcontroller board is connected to the USB port, the port name can be seen in the drop-down menu on the previous image.

If the Arduino IDE is used on Windows, port names are as follows:



For *Linux* users, for example port name is */dev/ttyUSBx*, where *x* represents integer number between *0* and *9*.

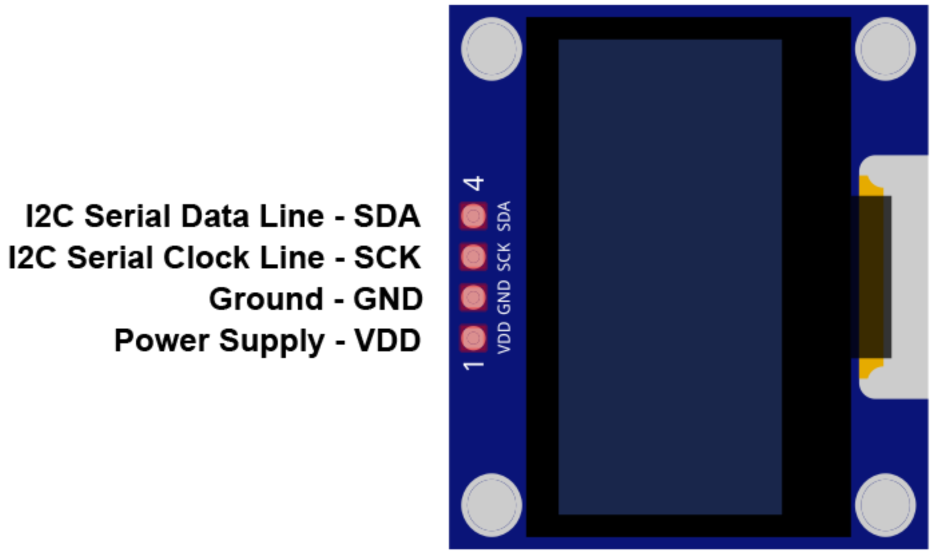
How to set-up the Raspberry Pi and Python

For the Raspberry Pi, first the operating system has to be installed, then everything has to be set-up so that it can be used in the *Headless* mode. The *Headless* mode enables remote connection to the Raspberry Pi, without the need for a *PC* screen Monitor, mouse or keyboard. The only things that are used in this mode are the Raspberry Pi itself, power supply and internet connection. All of this is explained minutely in the free eBook:
Raspberry Pi Quick Startup Guide

The *Raspbian* operating system comes with *Python* preinstalled.

The pinout

The 1.3 inch OLED screen has four pins. The pinout is shown on the following image:

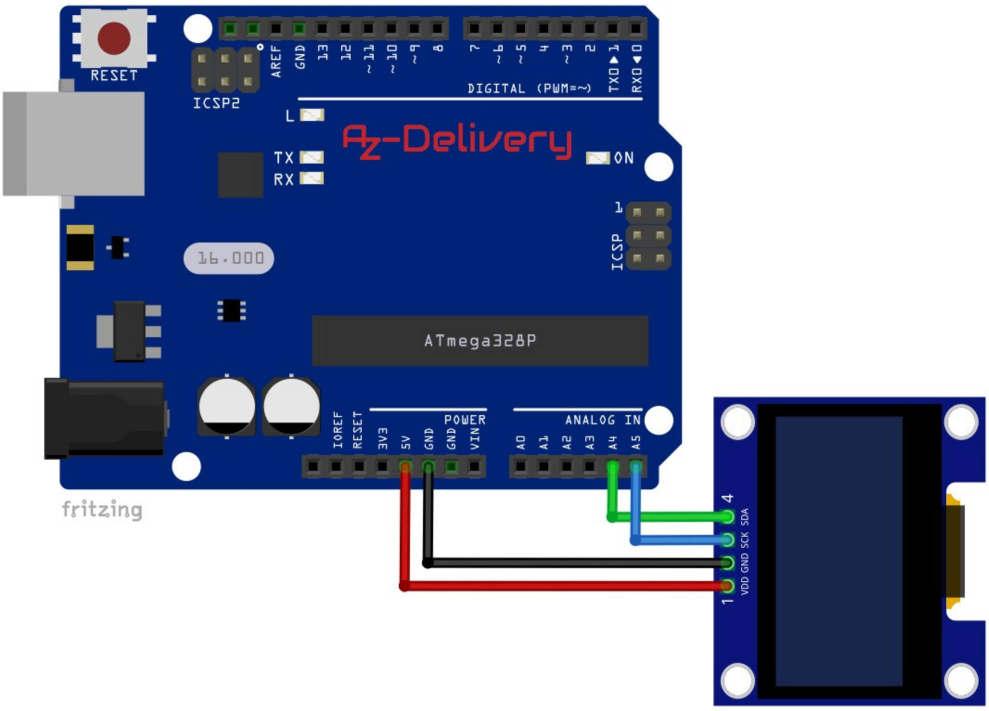


The screen has an on-board voltage regulator. The pins of the *1.3* inch OLED screen can be connected to *3.3V* or to *5V* power supply without danger to the sensor itself.

NOTE: When using Raspberry Pi, the power supply should be drawn from a 3.3V pin only.

Connecting the screen with Atmega328P Board

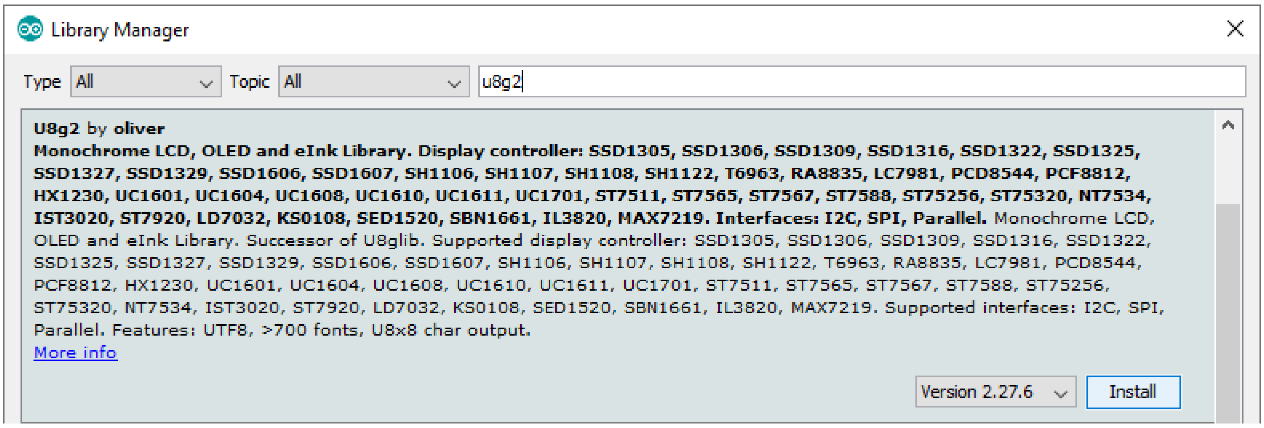
Connect the 1.3 inch OLED screen with the Atmega328P Board as shown on the following connection diagram:



Screen pin	Pin	Wire color
SDA	A4	Green wire
SCK	A5	Blue wire
GND	GND	Black wire
VCC	5V	Red wire

Library for Arduino IDE

To use the screen with an Atmega328P Board, it is recommended to download an external library for it. The library that is going to be used is called the *"U8g2"*. To download and install it, open Arduino IDE and go to: *Tools > Manage Libraries*. When a new window opens, type *"u8g2"* in the search box and install the library *"U8g2"* made by *"oliver"*, as shown in the following image:



Several sketch examples come with the library, to open one, go to:
File > Examples > U8g2 > full_buffer > GraphicsTest
With this sketch example, you can test your screen. However, the code used in the example is fairly complex. The sketch is modified to make a more beginner-friendly version of the code.

Sketch example

```
#include <U8g2lib.h>
#include <Wire.h>
#define time_delay 2000
U8G2_SH1106_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);
}
```

First, two libraries, the *U8g2lib* and *Wire* are imported. Next, an object called *u8g2* is created with the following line of code:

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

The created object represents the screen itself and it is used to control the screen. The *U8g2* library can be used for many other OLED screens, thus there are many constructors in the sketch examples from the library.

Next, a function called *u8g2_prepare()* is created, which has no arguments and returns no value. The five *u8g2* library functions are used.

The first function is called *setFont()*, which has one argument and returns no value. The argument represents the *u8g2* font. Follow the link to see the list of availale fonts:

https://github.com/olikraus/u8g2/wiki/fntlist8x8

The second function is called *setFontRefHeightExtendedText()*, which has no arguments and returns no value. It is used for drawing characters on the screen. For more detailed explanation, follow the link:
https://github.com/olikraus/u8g2/wiki/u8g2reference#setfontrefheightextendedtext

The third function is called *`setDrawColor()`*, which has one argument and returns no value. The argument value is an integer number which represents a color index for all drawing functions. Font drawing procedures use this argument to set the foreground color. The default value is *1.* If it is set to *0* then the space around the character is lit up, and the character is not. Argument value *2* can be used also, but there's no difference from *0*.

The fourth function is called *`setFontPosTop()`*, which has no argument and returns no value. This function controls the character position in one line of text. The function has a couple of versions. The first is *`setFontPosBaseLine()`* second is *`setFontPosCenter()`,* and third is *`setFontPosBottom()`*.Their purpose is to change the position of the characters in the one line.

The fifth function is called *`setFontDirection()`*, which has one argument and returns no value. The argument is an integer number which represents direction of the text. The value is an integer number in a range from *0* to *3* (*0* = *0*°, *1* = *90*°, *2* = *180*° and *3* = *270*°).

The function called *drawStr()* has three arguments and returns no value. It is used to display a constant string on the screen. The first two arguments represent the *X* and *Y* positions of the cursor, where the text is displayed. The third argument represents the text itself, a string value. Functions that set-up text layout should be used before using *drawStr()* function, or else the *drawStr()* function will use the default settings for the font, size and overall layout of the text.

To display shapes, specific library functions for each shape has to be used:

The function called *drawFrame()* has four arguments and returns no value. It is used to display frame, an empty rectangle. The first two arguments represent the *X* and *Y* positions of the top left corner of the frame. The third argument represents the width of the frame and the fourth argument represents the height of the frame.

The function called *drawRFrame()* has five arguments and returns no value. It is used to display a frame with rounded corners. The first two arguments represent the *X* and *Y* positions of the top left corner of the frame. The second two arguments represent the width and height of the frame and the fifth argument represents the corner radius.

The function called *drawBox()* has four arguments and returns no value. It is used to display a filled rectangle. The first two arguments represent the *X* and *Y* positions of the top left corner of the rectangle. The second two arguments represent the width and height of the rectangle.

The function called *drawRBox()* has five arguments and returns no value. It is used to display a filled rectangle with rounded edges. The first two arguments represent the *X* and *Y* positions of the top left corner of the rectangle. The second two arguments represent the width and height of the rectangle and the fifth argument represents the corner radius.

The function called *drawCircle()* has three arguments and returns no value. It is used to display a circle. The first two arguments represent the *X* and *Y* positions of the circle center point. The third argument represents the circle radius.

The function called *drawDisc()* has three arguments and returns no value. It is used to display a disc. The first two arguments represent *X* and *Y* positions of the disc center point. The third argument represents the disc radius.

The function called *`drawTriangle()`* has six arguments and returns no value. It is used to display a filled triangle. The first two arguments represent the *X* and *Y* positions of the first corner point of the triangle. The second two arguments represent the *X* and *Y* positions of the second corner point of the triangle. The last two arguments represent the *X* and *Y* positions of the last corner point of the triangle.

The function called *`drawLine()`* has four arguments and returns no value. It is used to display a line. The first two arguments represent the *X* and *Y* positions of the starting point of the line. The second two arguments represent *X* and *Y* positions of the end point of the line.

The function called *`drawUTF8()`* has three arguments and returns a value. It is used to display a text, the string value that can contain a character encoded as a *Unicode* character. The first two arguments represent the *X* and *Y* positions of the cursor and the third represents the text itself. The *Unicode* characters can be displayed in a couple of ways. The first is to copy and paste the existing character into the sketch, like in the following line of the code: `u8g2.drawUTF8(50, 20, "☂");`

The second is to create a *`char`* array, which has two values: the first value is a hexadecimal number of the *Unicode* character and the second value is a null character. This can be done by using the *`char`* array called *COPYRIGHT_SYMBOL*, like in the following lines of code:

```
const char COPYRIGHT_SYMBOL[] = {0xa9, '\0'};
```

```
u8g2.drawUTF8(95, 20, COPYRIGHT_SYMBOL); //COPYRIGHT SYMBOL
```

The third way of using the function is to use a hexadecimal number for the character itself, like in the following line of code:

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

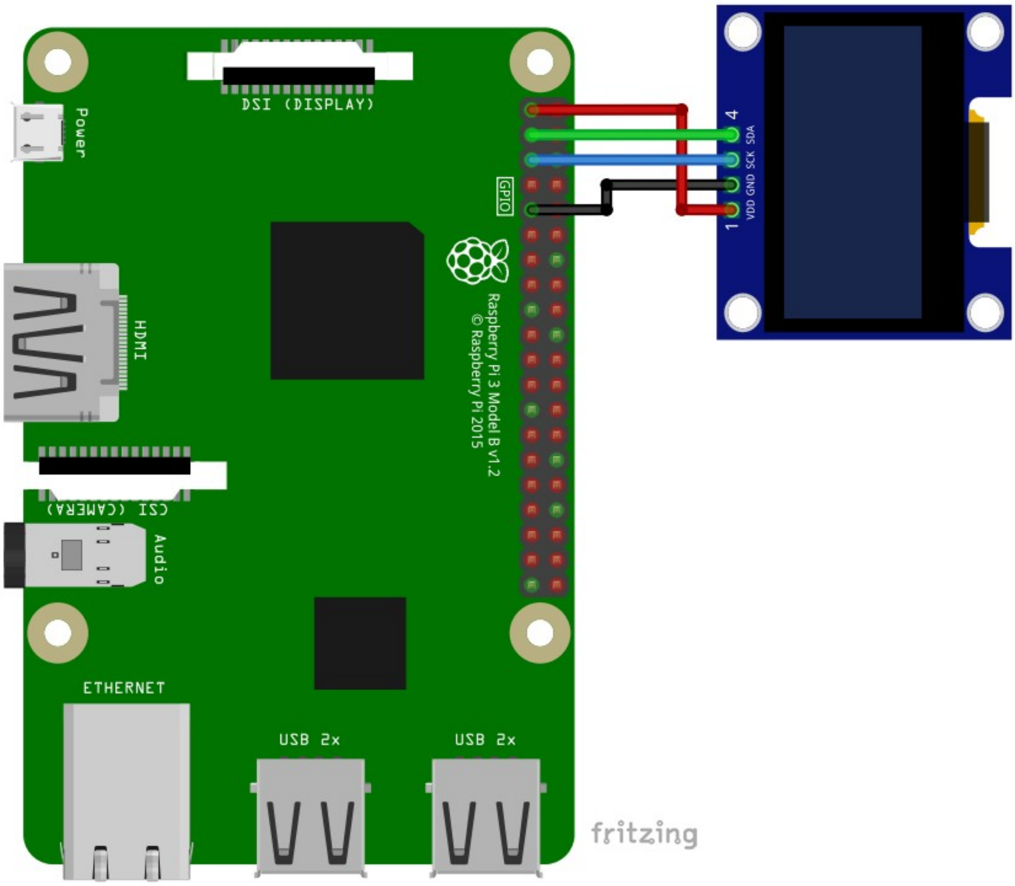
The function returns a value, an integer number which represents the width of the text (string).

To display something on the screen, the screen data buffer has to be cleared, then new value for data buffer has to be set (an image) and then to send the new value to the screen. This way, a new image will be displayed on the screen. In order to make this change visible, *delay()* function has to be used to delay the next change of the data buffer, like in the following lines of code:

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

Connecting the screen with Raspberry Pi

Connect the screen with the Raspberry Pi as shown on the following connection diagram:

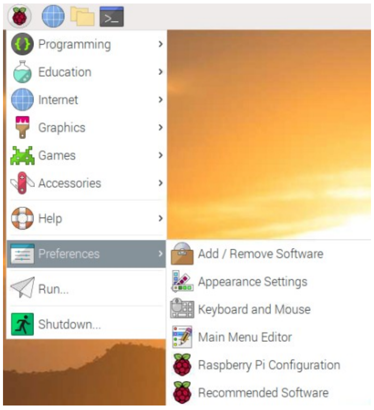


Screen pin	Raspberry Pi pin	Physical pin	Wire color
SDA	GPIO2	3	**Green wire**
SCL	GPIO3	5	**Blue wire**
GND	GND	9	**Black wire**
VCC	3V3	1	**Red wire**

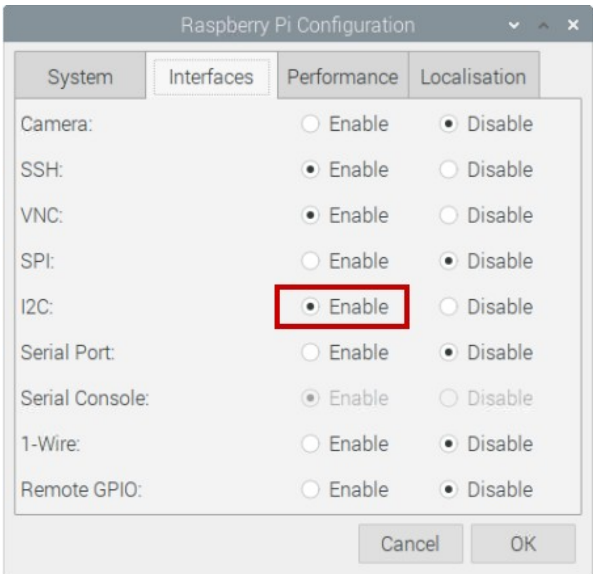
Enabling the I2C interface

In order to use the module with Raspberry Pi, I2C interface has to be enabled. Open following menu:

Application Menu > Preferences > Raspberry Pi Configuration



In the new window, under the tab *Interfaces*, enable the I2C radio button, as on the following image:



Finding the address of the OLED module

If it is enabled, we will use the command i2detect to find the module on the I2C bus: **i2cdetect -y 1**

The result should look like the one shown below:

```
pi@rpispy:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- 3c -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Our device was recognised with the address "0x3c". This is the standard address for this type of device.

Python-library

For the display of shapes, text and images we will use a Python library. On the current Raspberry Pi OS, Python3, pip3 and git are already pre-installed, but if this is not the case, you can install the whole thing with the following commands:

```
sudo apt-get install python3-dev libffi-dev libssl-dev python3-pil libjpeg-dev zlib1g-dev libfreetype6-dev liblcms2-dev libopenjp2-7 libtiff5 -y
```

```
sudo apt-get install python3-rpi.gpio python3-pip -y
```

```
sudo apt-get install git -y
```

We use the "luma.oled" library, which can be installed with the following command:

sudo -H pip3 install luma.oled

Python-Script

In the folder "pi" we now create a folder called "oled" and go to this folder

```
sudo mkdir oled
cd oled
```

luma.oled offers many examples and we can download them with the following command:

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

with:

cd luma.examples/examples/

we change to the folder in which the examples are located.

and with:

python3 demo.py

we can start one of the examples. If there is a white noise on the display, the correct controller must be passed. this can be done with:

```
python3 demo.py --device [controller]
```

luma.oled uses the SSD1306 by default, if you have SH1106 for example, starting the script would look like this:

```
python3 demo.py --device ssh1106
```

There are more examples in the folder.

With the command:

ls -a

you can see them.

Try out more examples.

And with:

sudo nano [examplescript]

the scripts can be edited.

Now it is the time to learn and make your own projects. You can do that with the help of many example scripts and other tutorials, which can be found on the Internet.

If you are looking for the high quality products for and Raspberry Pi, AZ-Delivery Vertriebs GmbH is the right company to get them from. You will be provided with numerous application examples, full installation guides, eBooks, libraries and assistance from our technical experts.

https://az-delivery.de

Have Fun!

Impressum

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