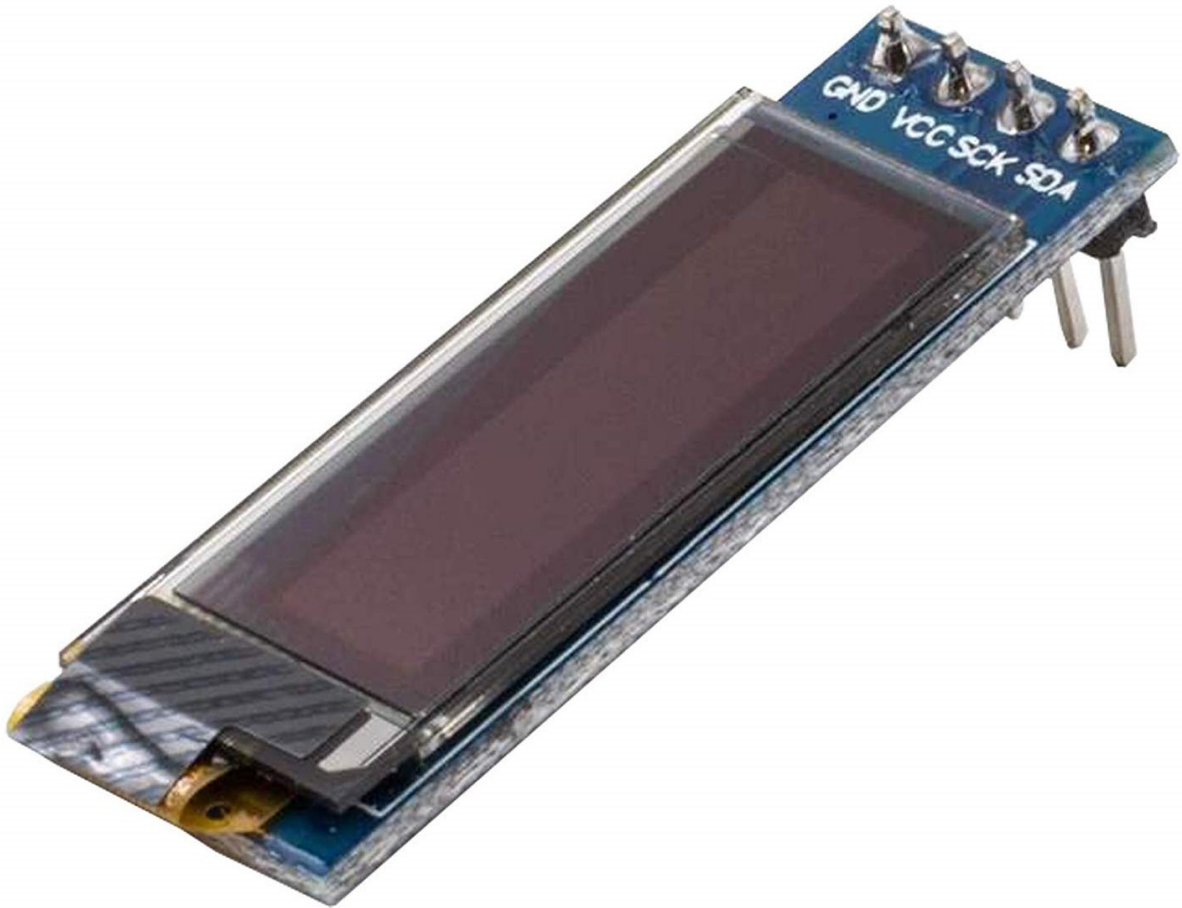


# AZ-Delivery

## Welcome!

Thank you for purchasing our *AZ-Delivery 0.91 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.....	3
Specifications.....	4
The pinout.....	5
How to set-up Arduino IDE.....	6
How to set-up the Raspberry Pi and the Python.....	10
Connecting the screen with Uno.....	11
Library for Arduino IDE.....	12
Sketch example.....	13
Connecting the screen with Raspberry Pi.....	24
Enabling the I2C interface.....	25
Libraries and tools for Python.....	26
Python script.....	31



## Introduction

OLED stands for Organic Light Emitting Diodes. OLED screens are arrays of LEDs stacked together in a matrix. The 0.91 OLED screen has a 128x32 pixels (LEDs). To control these LEDs we need a driver circuit or a chip. The screen has a driver chip called *SSD1306*. The driver chip has an I2C interface for communication with the main microcontroller. The I2C address of a driver chip is predefined with value *0x3C*.

The OLED screen and SSD1306 driver chip operate in the 3.3V range. But there is an on-board 3.3V voltage regulator, which means that these screens can operate in the 5V range.

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

## Specifications

- » Operating voltage range: from 3.3V to 5V DC
- » Communication interface: I2C
- » Pixel color: White
- » Operating temperature: from -20 to 70 °C
- » Low power consumption: less then 8mA
- » Dimensions: 30 x 12 x 2mm [1.2 x 0.5 x 0.1in]

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

## The pinout

The 0.91 inch OLED screen has four pins. The pinout is shown on the following image:



The screen has an on-board 3.3V voltage regulator. The pins of the 0.91 inch OLED screen can be connected to both 3.3V or 5V logic and power supply without danger to the screen itself.

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

## 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.

### Download the Arduino IDE



The screenshot shows the Arduino IDE download page. On the left, there is a teal circle with a white infinity symbol containing a minus and a plus sign. To its right, the text reads: **ARDUINO 1.8.9**, followed by a description of the IDE as open-source software written in Java, and a note that it can be used with any Arduino board. On the right side, there is a teal sidebar with links for different operating systems: Windows (installer and ZIP file), Windows app (with a 'Get' button), Mac OS X (10.8 Mountain Lion or newer), and Linux (32 bits, 64 bits, ARM 32 bits, and ARM 64 bits). At the bottom of the sidebar are links for Release Notes, Source Code, and Checksums (sha512).

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

# Az-Delivery

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`.

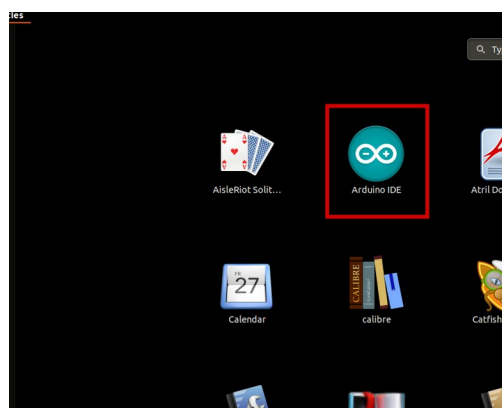
To run the first script in the terminal, open the terminal in the extracted directory and run the following command:

```
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.



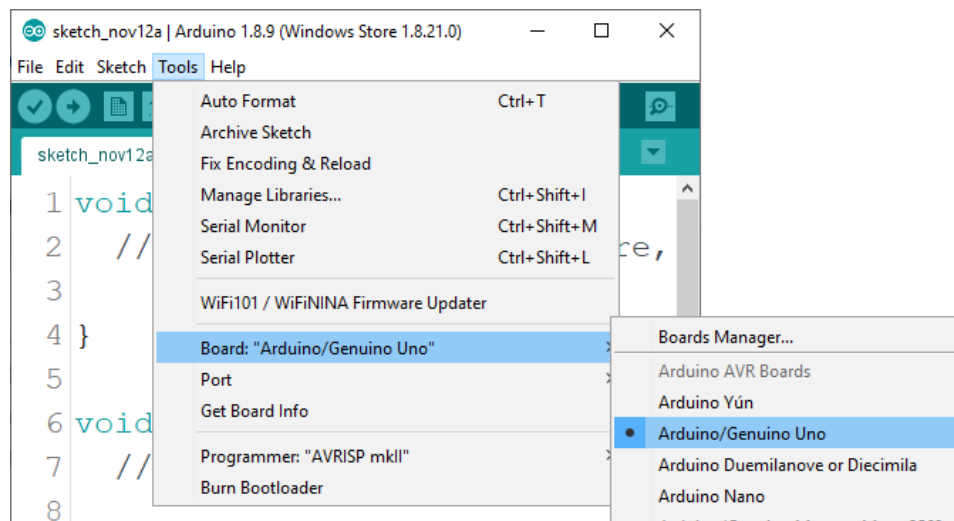
# Az-Delivery

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 an Arduino 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:



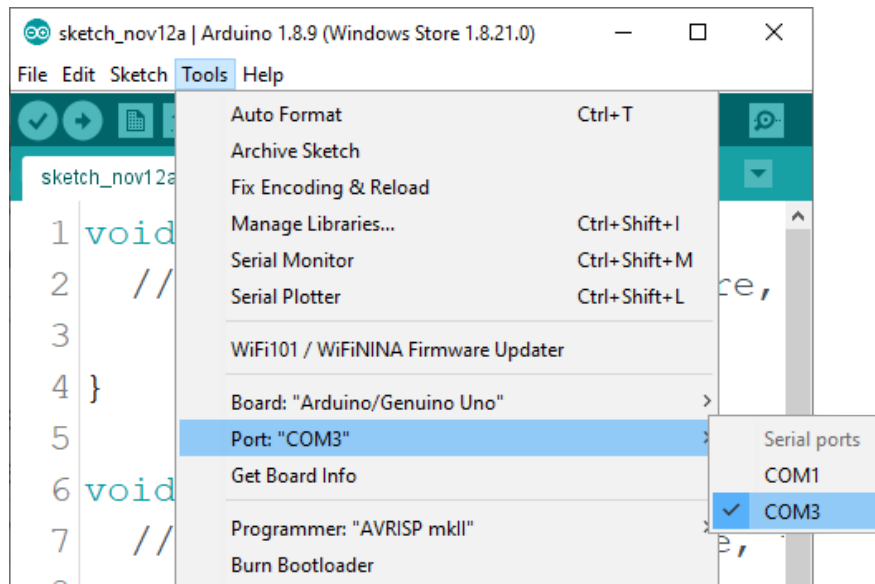
The port to which the Arduino board is connected has to be selected. Go to:

*Tools > Port > {port name goes here}*

and when the Arduino 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

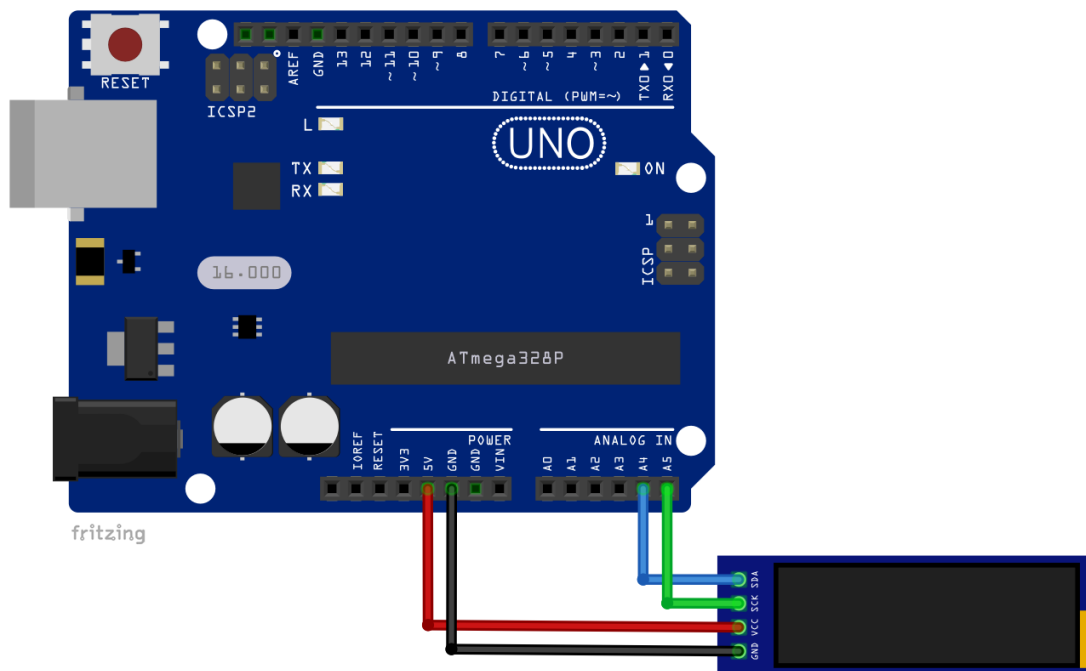
For the Raspberry Pi, first the operating system has to be installed, then everything has to be set-up so that it can be used in the *Headless* mode. The *Headless* mode enables remote connection to the Raspberry Pi, without the need for a *PC* screen Monitor, mouse or keyboard. The only things that 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.

## Connecting the screen with Uno

Connect the 0.91 inch OLED screen with the Uno as shown on the following connection diagram:



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

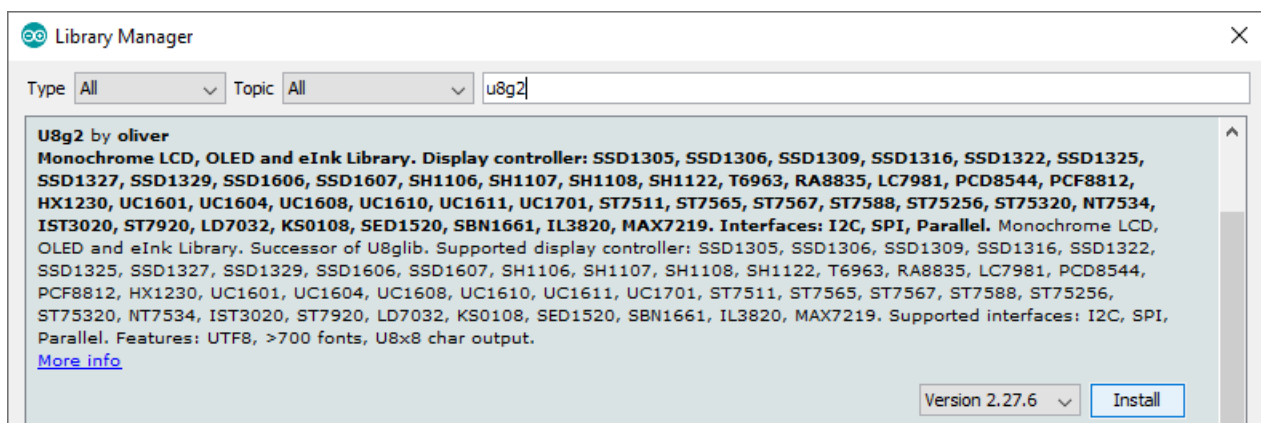
# Az-Delivery

## Library for Arduino IDE

To use the screen with Uno, it is recommended to download an external library for it. The library that is used in this eBook 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*

The screen can be tested with this sketch. In this eBook, the sketch code is modified in order to create more beginner friendly version of code.

# Az-Delivery

## Sketch example

```
#include <U8g2lib.h>
#include <Wire.h>
#define time_delay 2000
U8G2_SSD1306_128X32_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);
}
```

# Az-Delivery

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

# Az-Delivery

```
#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 };
```

# Az-Delivery

```
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);  
}
```



# Az-Delivery

```
// 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);
}
```

# Az-Delivery

At the beginning of the sketch two libraries are imported the *U8g2lib* and *Wire*.

Next, 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.

After that, the function called *u8g2\_prepare()* is created, which has no arguments and returns no value. Inside this function, 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. The list of available fonts can be found on the following [link](#).

The second function is called *setFontRefHeightExtendedText()* which has no arguments and returns no value. It is used for drawing characters on the screen. More detailed explanation of this function can be found on the following [link](#).

# Az-Delivery

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 also be used, but there is 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 the text. The function has a couple of versions. The first is *setFontPosBaseLine()* second is *setFontPosCenter()*. The third is *setFontPosBottom()* and 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 the range of *0* to *3*, where *0* =  $0^\circ$ , *1* =  $90^\circ$ , *2* =  $180^\circ$  and *3* =  $270^\circ$ .

# Az-Delivery

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* position of the cursor, where the text is displayed. The third argument represents the text itself, a constant string value. The functions that set text layout before using *drawStr()* function should be used, otherwise the *drawStr()* function uses default settings for the font, size and overall layout of the text.

To display shapes, specific functions for each shape are used:

The function called *drawFrame()*, has four arguments and returns no value. It is used to display a frame, an empty rectangle. The first two arguments represent the *X* and *Y* position 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* position 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.

# Az-Delivery

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* position of the top left corner of the rectangle. The second two arguments represent the width and height of the rectangle, respectively.

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* position of the top left corner of the rectangle. The second two arguments represent the width and height of the rectangle, respectively. 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* position of the disc center point. The third argument represents the disc radius.

# Az-Delivery

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* position 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* position of the starting point of the line. The second two arguments represent *X* and *Y* position 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 which may contain a character encoded as a *Unicode* character. The first two arguments represent the *X* and *Y* position 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 (“\0”). This can be done by using the *char* array called *COPYRIGHT\_SYMBOL*, like in the following lines of the code:

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

# Az-Delivery

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
```

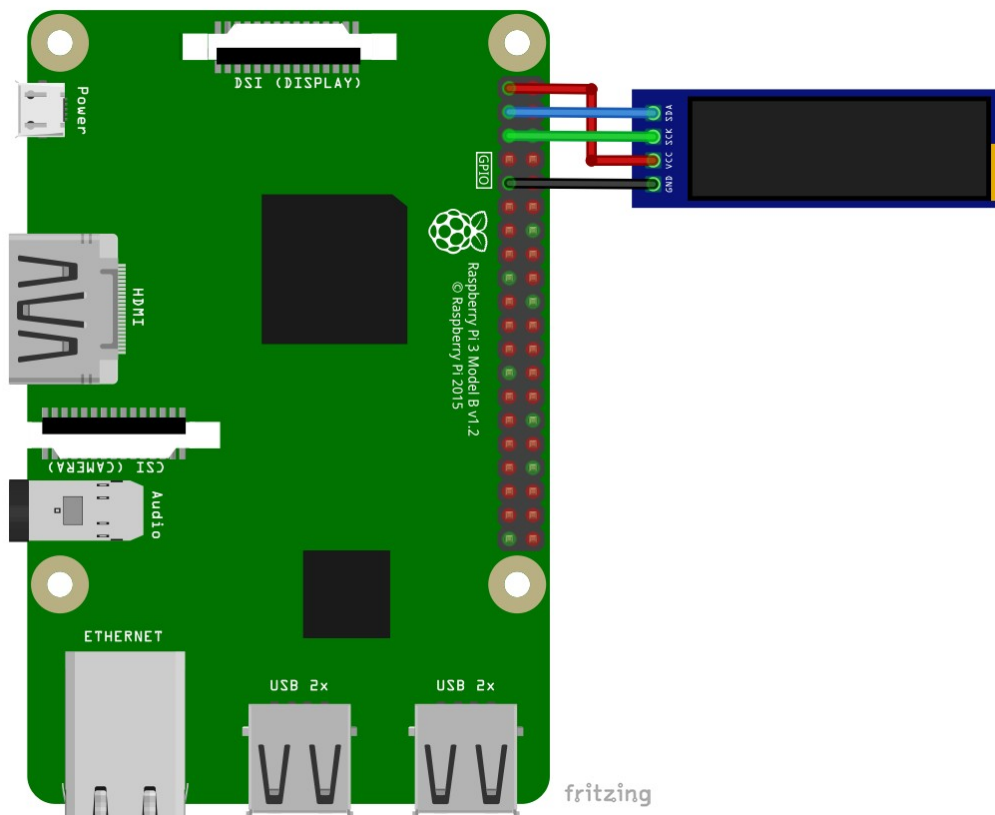
The function returns a value, an integer number which represents the width of the text (*string*).

To display something on the screen, the screen data buffer has to be cleared first, then a new value is set (an image) for data buffer, then a new value of data buffer is send to the screen. This way, a new image is displayed on the screen. In order to see this change, *delay()* function has to be used to shift 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 No.	Wire color
SDA	GPIO2	3	Blue
SCK	GPIO3	5	Green
VCC	3V3	1	Red
GND	GND	9	Black



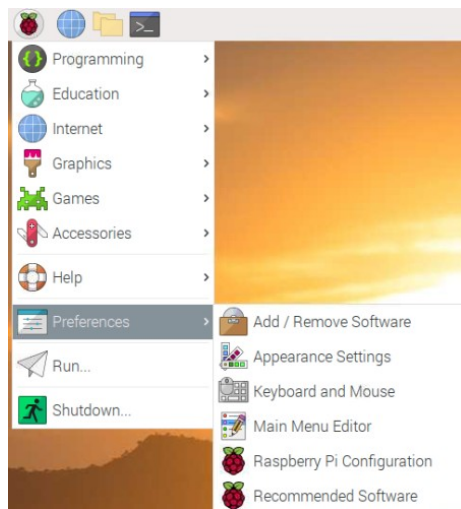
# Az-Delivery

# Az-Delivery

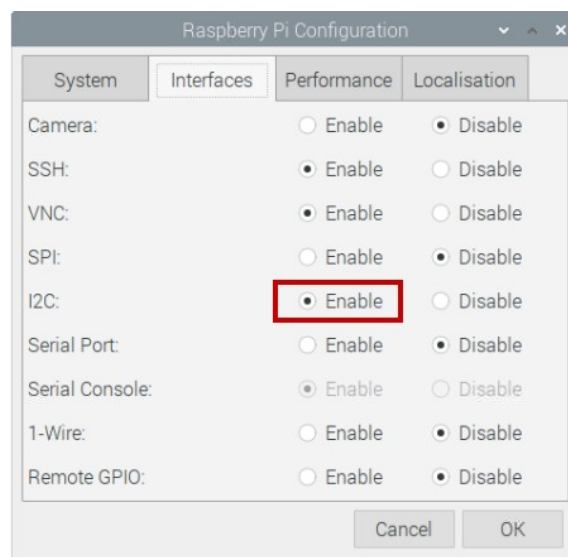
## Enabling the I2C interface

In order to use the screen 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:





## Libraries and tools for Python

In order to use the screen with the Raspberry Pi, it is recommended to download and install an external library. The library which will be used is called *Adafruit\_Python\_SSD1306*.

Before installing the main library, several tools and libraries have to be installed first.

First, the *python-smbus* and *i2c-tools* had to be installed.

Open the terminal and run the following commands:

```
sudo apt-get update
```

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

The following list are tools required to be on the system:

*python3-dev, python-imaging, python3-pil, python3-pip, python3-setuptools, python3-rpi.gpio*

If tools from the list are not present, they can be installed by running the following commands from the terminal window:

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

# Az-Delivery

In order to download the library the *git* has to be installed. If the *git* is not present on the system, installation can be done with the following commands:

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

The library repository can be cloned by running following command:

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

The directory has to be changed to the *Adafruit\_Python\_SSD1306*, by running the following command: **cd Adafruit\_Python\_SSD1306**

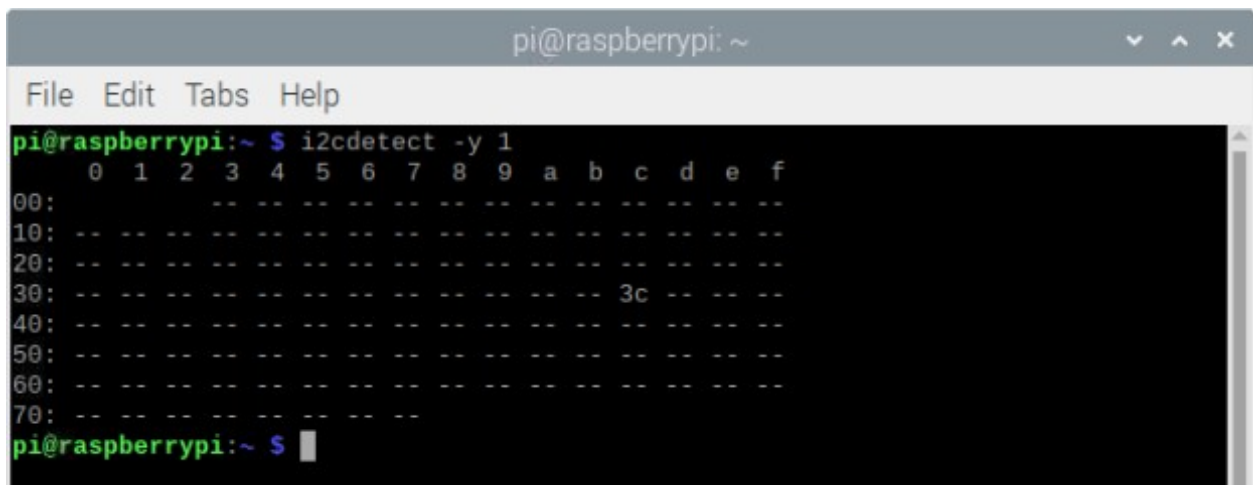
Then, library will be installed by running the following command:

```
sudo python3 setup.py install
```

# Az-Delivery

Before using any device connected to the I2C interface, I2C address has to be detected first. To detect the screen I2C address, following command should be run in the terminal: **i2cdetect -y 1**

The result should look like the following image:



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~$ 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:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
pi@raspberrypi:~$
```

Where 0x3c is the I2C address of the screen.

If I2C interface is not enabled with the previous command, the error will appear like on the the following image:



```
pi@raspberrypi: ~/Scripts  
File Edit Tabs Help  
pi@raspberrypi:~/Scripts$ i2cdetect -y 1  
Error: Could not open file '/dev/i2c-1' or '/dev/i2c/1': No such file or directory  
pi@raspberrypi:~/Scripts$
```

# Az-Delivery

There are several script examples that comes with the library, navigate to the directory: `/Adafruit_Python_SSD1306/examples`

by running the following command:

```
cd ~/Adafruit_Python_SSD1306/examples
```

This directory contains several script examples, including:

*shapes.py*,

*image.py*,

*stats.py*

and others.

Focus is on the script *shapes.py*. To run the script, open terminal in the directory where the script is saved and run the following command:

```
python3 shapes.py
```

# Az-Delivery

## Python script

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

disp = Adafruit_SSD1306.SSD1306_128_32(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!]\n')
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)
```

# Az-Delivery

```
# 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)
```



# Az-Delivery

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

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

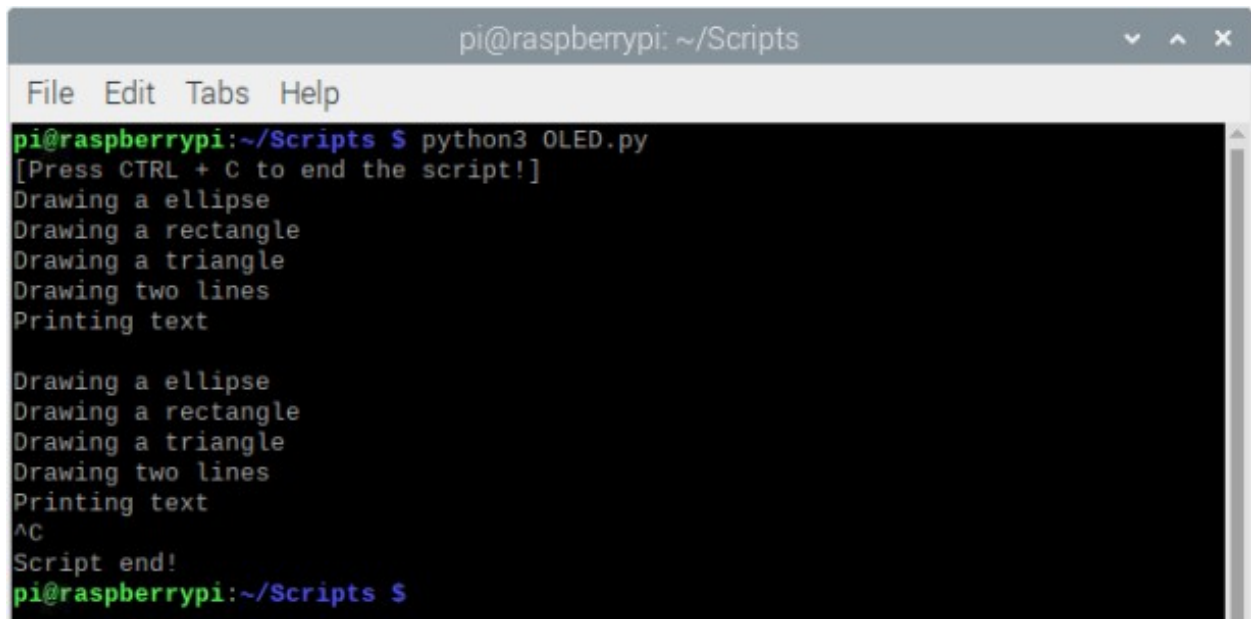
finally:
    disp.clear()
    disp.display()
```

# Az-Delivery

Save the script by the name *OLED.py*. To run the script, open terminal in the directory where the script is saved and run the following command:

**python3 OLED.py**

The result should look like the output on the following image:

A screenshot of a terminal window titled 'pi@raspberrypi: ~/Scripts'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows the command 'python3 OLED.py' being executed. The output consists of two identical blocks of text: 'Drawing a ellipse', 'Drawing a rectangle', 'Drawing a triangle', 'Drawing two lines', and 'Printing text'. After the second block, the user presses 'Ctrl+C', indicated by '^C' on the screen, and the terminal displays 'Script end!'. The prompt 'pi@raspberrypi:~/Scripts \$' is visible at the bottom.

```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 OLED.py
[Press CTRL + C to end the script!]
Drawing a ellipse
Drawing a rectangle
Drawing a triangle
Drawing two lines
Printing text

Drawing a ellipse
Drawing a rectangle
Drawing a triangle
Drawing two lines
Printing text
^C
Script end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL + C* on the keyboard.

# Az-Delivery

The script starts with importing several libraries and functions.

Next, object called *disp* is created with the following line of code:

```
disp = Adafruit_SSD1306.SSD1306_128_32(rst=None)
```

Where *rst=None* is used. This represents the reset pin, which the 0.91 inch OLED screen does not have. The *Adafruit\_SSD1306* library can be used for many other OLED screens that is why there is option for this pin.

The object *disp* represents the screen itself and this object is used to send commands to the screen.

Next, the screen object is initialized, the screen data buffer has been cleared. After this, images are created.

To create an image, first empty image has to be made with the dimensions of the screen. It is done with the following line of code:

```
image = Image.new('1', (disp.width, disp.height))
```

Next, with this *image* object the *draw* object is created, which is used to draw shapes: `draw = ImageDraw.Draw(image)`

Then, *try-except-finally* block of code is created. In the *try* block of code the indefinite loop is created (*while True:*). In the indefinite block of code there is the algorithm for controlling the screen.

# Az-Delivery

The *except* block of code executes when *CTRL + C* is pressed on the keyboard. This is called keyboard interrupt and it is used to end the script. When this block code is executed, the message *Script end!* Is displayed in the terminal.

The *finally* block of code is executed at the end of the script execution. It is used to clear the data buffer of the screen and to disable all used GPIO pin modes and/or interfaces.

To draw the rectangle, the *rectangle()* function is used. The function accepts three arguments and returns no value. The first argument is a *tuple* of four elements, where the first two elements are *X* and *Y* position of the top right corner of the rectangle. The third element is the width of the rectangle and the fourth is the height of the rectangle. The second argument is the *outline* argument and the third argument is the *fill* argument.

The *X* position value starts at the left side of the screen (value of *0*) and ends at the right side of the screen (value of *127*). The *Y* position value starts at the top side of the screen (value of *0*) and ends at the bottom side of the screen (value of *31*).

# Az-Delivery

The *outline* argument represents the color of the shape edge and the *fill* argument represents the color of the shape itself. Because OLED screens are used, colors are black and white, black - the pixel is turned *OFF*, and the white - the pixel is turned *ON*. When value of zero is saved to the *outline* or *fill* argument this means that it is a black color. When any other value higher than zero is saved, for example 255, this represents white color.

To draw ellipse or circles, the *ellipse()* function is used. The function accepts three arguments and returns no value. The first argument is a *tuple* of four elements. The first two elements represent the *X* and *Y* positions of the top right corner of the rectangle that contains the ellipse. The third element is the width of the rectangle and the fourth element is the height of the rectangle. If the width is equal to the height the shape that is drawn is the circle. The second argument is the *outline* argument, and the third argument is the *fill* argument.

To draw a polygon, the *polygon()* function is used. A polygon is a triangle, a rectangle, or any other shape with 3 or more corners. The function accepts three arguments. The first argument is a *list* of three or more *tuples*. The *tuples* have two elements and represents one corner point. Elements in the *tuple* represent the *X* and *Y* position of the shape corner point. The number of *tuples* in the *list* is arbitrary, three or more *tuples*, which depends on what shape is drawn. The second argument is the *outline* argument and the third argument is the *fill* argument.

# Az-Delivery

To draw a line, the `line()` function is used. The function accepts two arguments and returns no value. The first argument is a *tuple* of four elements, where the first two elements represent *X* and *Y* position of the starting point of a line and the second two elements represent *X* and *Y* position of the end point of a line. The second argument is the *fill* argument.

To display the text, the `text()` function is used. The function accepts four arguments and returns no value. The first argument is a *tuple* of two elements, which represents *X* and *Y* positions of the cursor where the text is displayed. The second argument represents the text itself, (a *string* value). The third argument is the *font* argument, and the fourth argument is the *fill* argument. The *font* argument represents the used font. To set the value of the *font* argument the following line of code is used with a default library font: `font = ImageFont.load_default()`

There is a option to use different fonts, but that is be covered it in this eBook.



Now is the time to learn and make projects on your own. 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 Arduino 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>