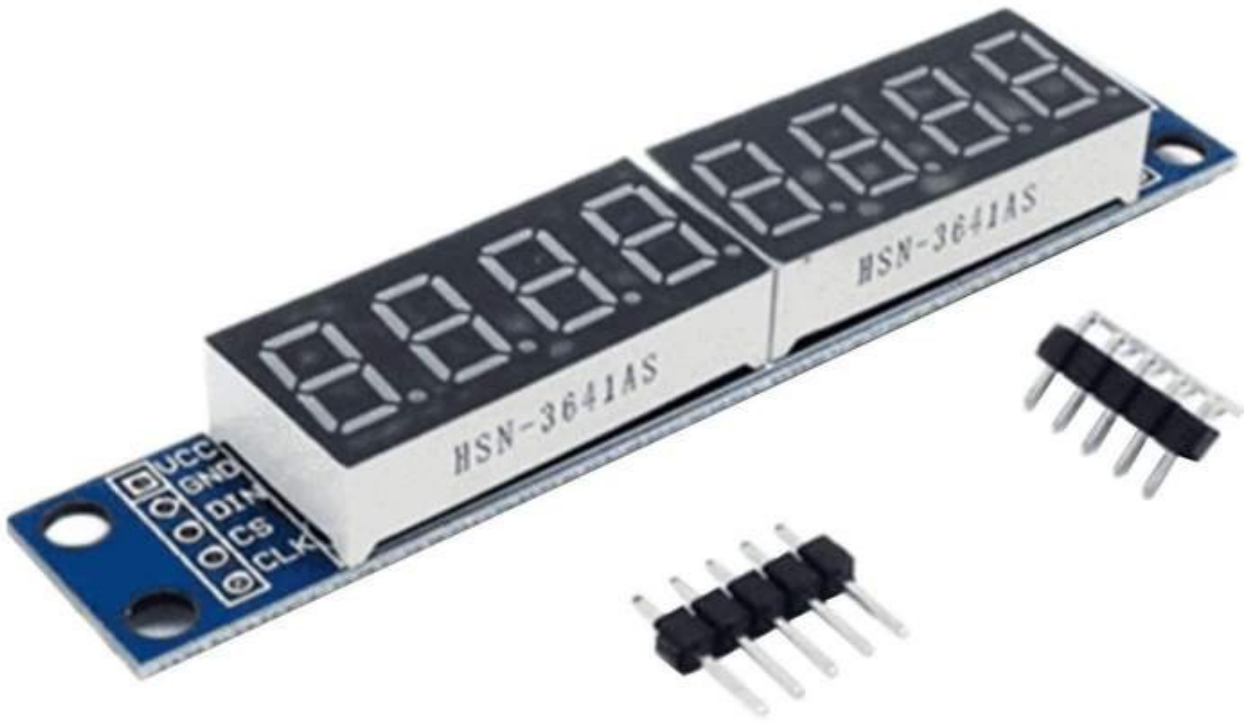
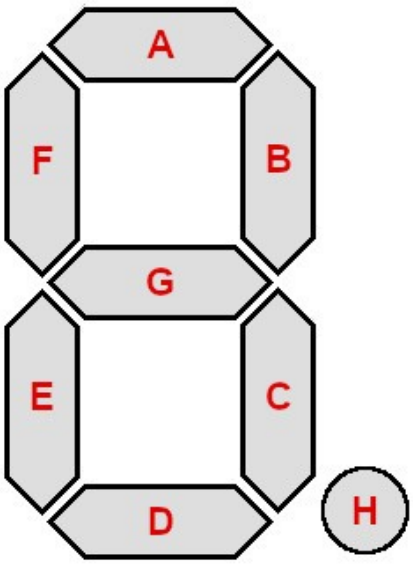
Welcome!

Thank you for purchasing our *AZ-Delivery MAX7219 8 bit Seven Segment Display Module*. On the following pages, we will introduce you to how to use and set-up this handy device.

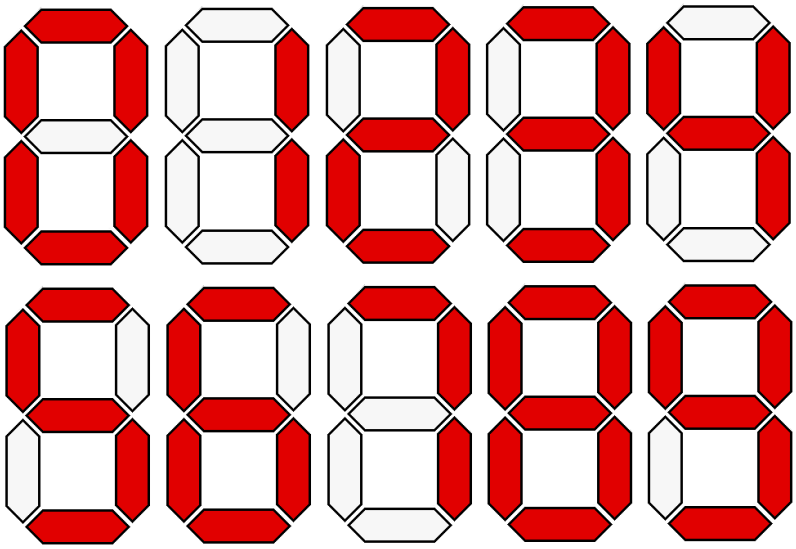
Have fun!



Seven segment screens are actually a screens with seven segments arranged in form of number 8. More accurately these screens have 8 segments, where the eighth segment is a dot near digital number 8. Every segment has its own name, labeled with letters from A to G (to H):



By turning *ON* specific segments (LEDs), digital numbers or characters are created:



In order to control one segment or LED, two pins are required, one for power and the other for ground. All ground pins are connected together, and other pins (or LED control pins), are separated. This means that seven segment screens have *8* control pins and one ground pin (*9* pins in total). To control these screens, *8* digital pins of microcontroller are required, which can be impractical. If you want to control more than one of the seven segment screen, you would need too many pins, which most microcontrollers do not have. That is why we are using driver chips, an integrated circuit dedicated for driving these kind of screens. In our case, we are using driver chip called "*MAX7219*". The driver chip is designed to drive screens made of many LEDs. The driver chip is capable to drive one or up to *8* of the seven segment screens. The first advantage of the driver chip is that only *4* digital pins enable communication with it, and drive *8* of the seven segment screens. The second advantage is that you can connect multiple driver chips serially, and use the same 4 pins to drive more than *8* of the seven segment screens.

SPI interface enables communication between microcontroller and MAX7219 driver chip. SPI stands for Serial Peripheral Interface, and it is used for sending and receiving data between a microcontroller and other peripherals (sensors, memories, etc.). In some sources it is called 4-wire protocol, because only 4 wires are needed to make hardware connections (plus ground wire). Usually, communication is established between one master and one or many slave devices.

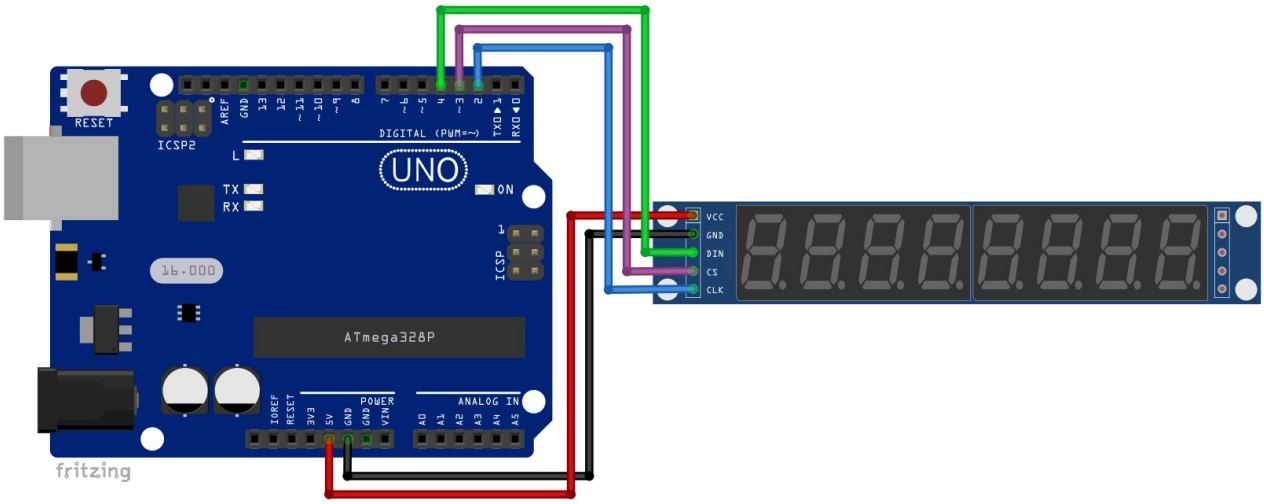
Specifications:

»	Power supply and logic voltage range:	from 3.3V to 5V DC
»	Driver chip:	MAX7219
»	Interface Type:	SPI interface
»	Maximum Clock:	10MHz
»	Number of segments:	8
»	Digit LED Color:	Bright RED
»	Brightness control:	16 levels
»	Cascading Ability:	Yes, Daisy chain
»	Dimensions:	84 x 16mm [3.3 x 0.6in]

It is possible to use the same 4 wires to control multiple 8 bit seven segment modules. This is called “Daisy chaining” of the modules via the same SPI interface. All we have to do is to connect the next module on the output pins of the first module. But be careful, more modules require more power, for that matter, use external power supply, rather than Arduino board voltage regulator.

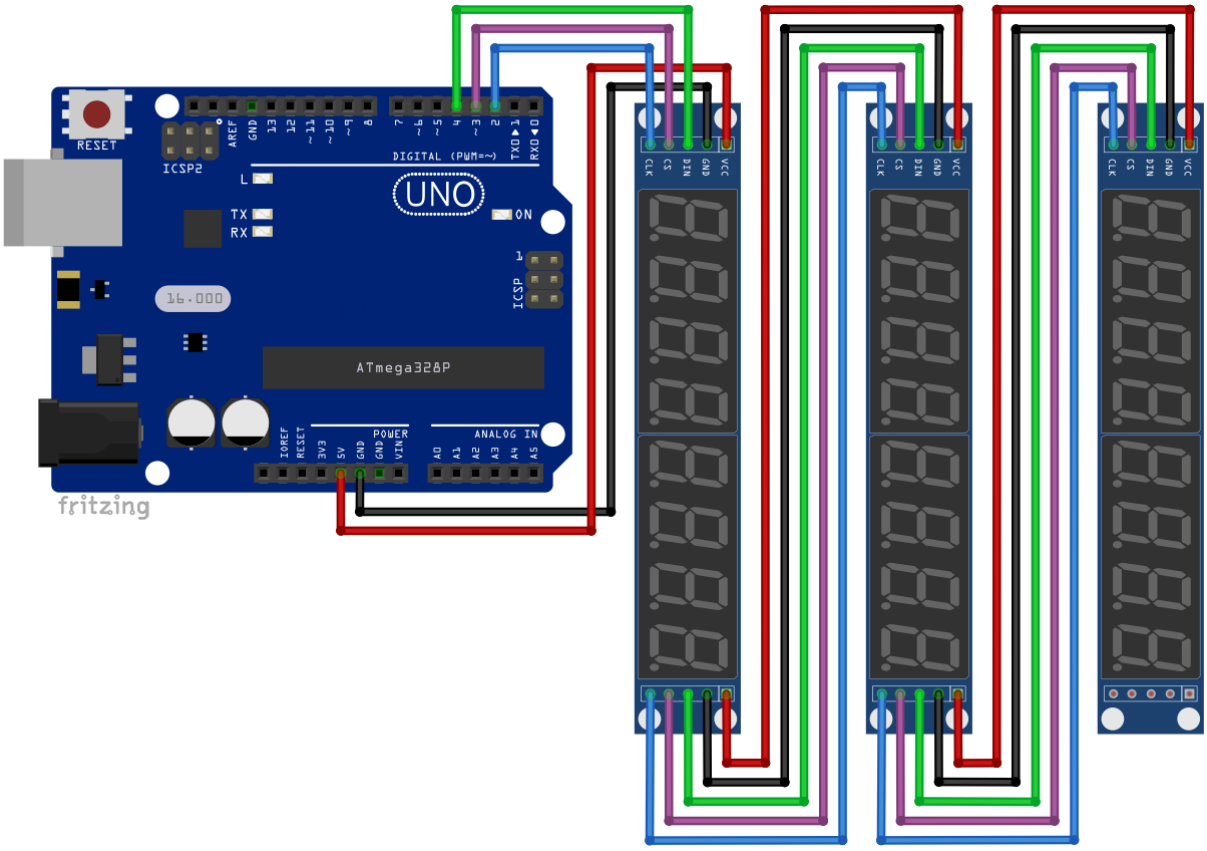
Connecting the module with Uno

Connect the module with the Uno as shown on the connection diagram below:



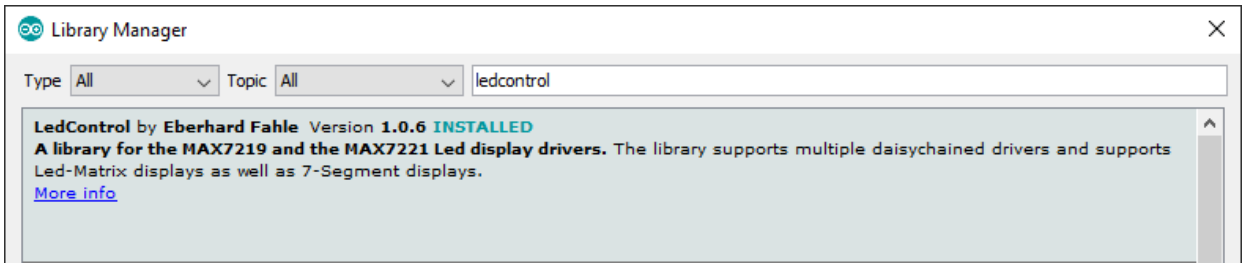
Module pin	>	Uno pin	
VCC	>	5V	**Red wire**
GND	>	GND	**Black wire**
DIN	>	D4	**Green wire**
CS	>	D3	**Purple wire**
CLK	>	D2	**Blue wire**

Down below is and example of the "*Daisy chaining*" connection diagram:



Arduino IDE library

In order to use the module with the Uno, it is best if we download a library for it. Open the Arduino IDE and go to *Tools* > *Manage Libraries*, and a new window will open. Type "*ledcontrol*" in the search box and install a library "*LedControl*" made by *"Eberhard Fahle"* like on the image below:



With this library comes three sketch examples. We will use and modify the sketch called “*LCDemo7Segmet*”:
File > *Examples* > *LedControl* > *LCDemo7Segmet*.

Sketch code:

```
#include "LedControl.h"
// DIN = 4, CS = 2, CLK = 3, Number of MAX7219 chips = 1
LedControl lc = LedControl(4, 2, 3, 1);

void setup() {
  // The MAX7219 chip is in power-saving mode on startup
  lc.shutdown(0, false);  // wakeup call
  lc.setIntensity(0, 8);  // Set brightness level (0 is min, 15 is max)
  lc.clearDisplay(0);     // Clear display register
}
void loop() {
  for(int i = 0; i < 21; i++) {
    lc.setChar(0, i, 'a', false);       //  a
    lc.setDigit(0, i - 1, 2, false);    //  z
    lc.setRow(0, i - 2, 0x01);          //  -
    lc.setChar(0, i - 3, 'd', false);   //  d
    lc.setChar(0, i - 4, 'e', false);   //  e
    lc.setRow(0, i - 5, B00001110);     //  l
    lc.setDigit(0, i - 6, 1, false);    //  i
    lc.setRow(0, i - 7, B00111110);     //  v
    lc.setChar(0, i - 8, 'e', false);   //  e
    lc.setRow(0, i - 9, 0x05);          //  r
    lc.setRow(0, i - 10, B00111011);    //  y
    lc.setChar(0, i - 11, ' ', false); //  space (empty)
    delay(150);
  }
}
```

At the beginning of the sketch we include *LedControl* library and create and initialize screen object called "*lc*". The screen object is initialized in the following line of the code:

```
LedControl lc = LedControl(4, 2, 3, 1)
```

We call the constructor `LedControl()` that accepts four arguments. First three arguments represent connections of the module pins with Uno pins:

Module pins	>	Uno pins
DIN (MOSI)	>	D4
CS (SS)	>	D3
CLK	>	D2

The fourth argument is the number of used MAX7219 driver chips. We used one in our sketch. But if you are connecting multiple *8* bit seven segment screen modules (Daisy chaining), you have to specify the number of modules in this fourth argument.

In the `setup()` function, first we have to turn *ON* the module. The MAX7219 driver chip is in the power-saving mode when it is not used, and to turn it *ON* we have to make a "*wakeup call*". We do it with the following line of the code: `lc. shutdown(0, false)`

The first argument value "*0*" is the number of the MAX7219 chip. When you have multiple modules connected in a Daisy chain, you have to wake up all the MAX7219 chips. So the value of the first argument will be the number of MAX7219 in array of modules, starting from *0*. Second argument "`false`" is used to wake up the module. If you change it to "`true`", this will put the module into power-saving mode.

Second thing we have to do in the *setup()* function is to set the brightness level of the seven segment screens. We do this with the following line of the code: `ls.setIntensity(0, 8)`
The first argument "*0*" is the number of the module in the Daisy chain, and the second argument "*8*" is the level of brightness. Minimum level of brightens is *0* - turned *OFF*, and maximum level is *15*.

Finally, we clear the screen with the following line of the code:
`lc.clearDisplay(0)`
The one argument "*0*" is the number of the module in the Daisy chain.

In the *loop()* function we scroll the message "*AZ-Delivery*" on the module. To do this we use several functions defined in the *LedControl* library. To output specific number or character on the module, we use the following function: `lc.setChar(chip, position, character, point)`
The first argument is the number of the module in the Daisy chain. The second argument is the character position on the module. The module has *8* of seven segment screens, so position values range from *0* to *7*.

The third argument is a constant of type *char*, a character or a number, and its values are following characters:

- Single digit numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9
- Letters:
  - "A" or "a" - displays upper case
  - "B" or "b" - displays lower case
  - "C" or "c" - displays lower case

- “D” or “d” - displays lower case
- “E” or “e” - displays upper case
- “F” or “f” - displays upper case
- “H” or “h” - displays upper case
- “L” or “l” - displays upper case
- “P” or “p” - displays upper case
- “-” - displays the dash sign (minus sign)
- “.” - turn *ON* the decimal-point, H segment
- “_” - displays the underscore
- <SPACE> - the blank or space char

The fourth argument is a point, or the *H* segment of the seven segment screen. You can turn it *OFF* with *point* = *false*, or turn it *ON* with *point* = *true*. For example: `lc.setChar(0, 0, ‘a’, false)`
which outputs letter ‘*a*’ on the first seven segment screen of the module, with *H* segment turned *OFF*.

Similar function is: `lc.setDigit(chip, position, number, point)`
All arguments are the same as in the *`setChar()`* function, except the third argument. The third argument is a constant integer number which values are in the range from *0* to *9*. For example:
`lc.setDigit(0, 0, 2, false)`
which outputs number *2* on the second seven segment screen of the module, with *H* segment turned *OFF*.

And last function used in the sketch is *setRow()*. This function accepts three arguments. The first is the number of the module in the Daisy chain. The second argument is the character position on the module, like in the *setChar()* or *setDigit()* functions. And the third argument is hexadecimal or binary number, which value represents the character that will be displayed on the module. The binary value is represented like this: *BHABCDEFG*

where the first "*B*" represents binary number, and characters "*HABCDEFG*" are the segment letters of the specific seven segment screen. To turn *ON* specific segment, for example *B* segment, value of this binary number is *B00100000*, or if we want to output number *6*, the value is *B01011111*.

We can convert this binary value into hexadecimal (for number *6*) like this:

```
B 0101 1111
```

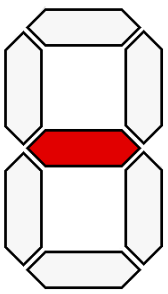
split the *8* bit binary number into two *4* bit numbers and convert these values into hexadecimal value, so we get two digit hexadecimal value of:

0x5f

In the sketch we use this function to display a dash sign "-", like this:

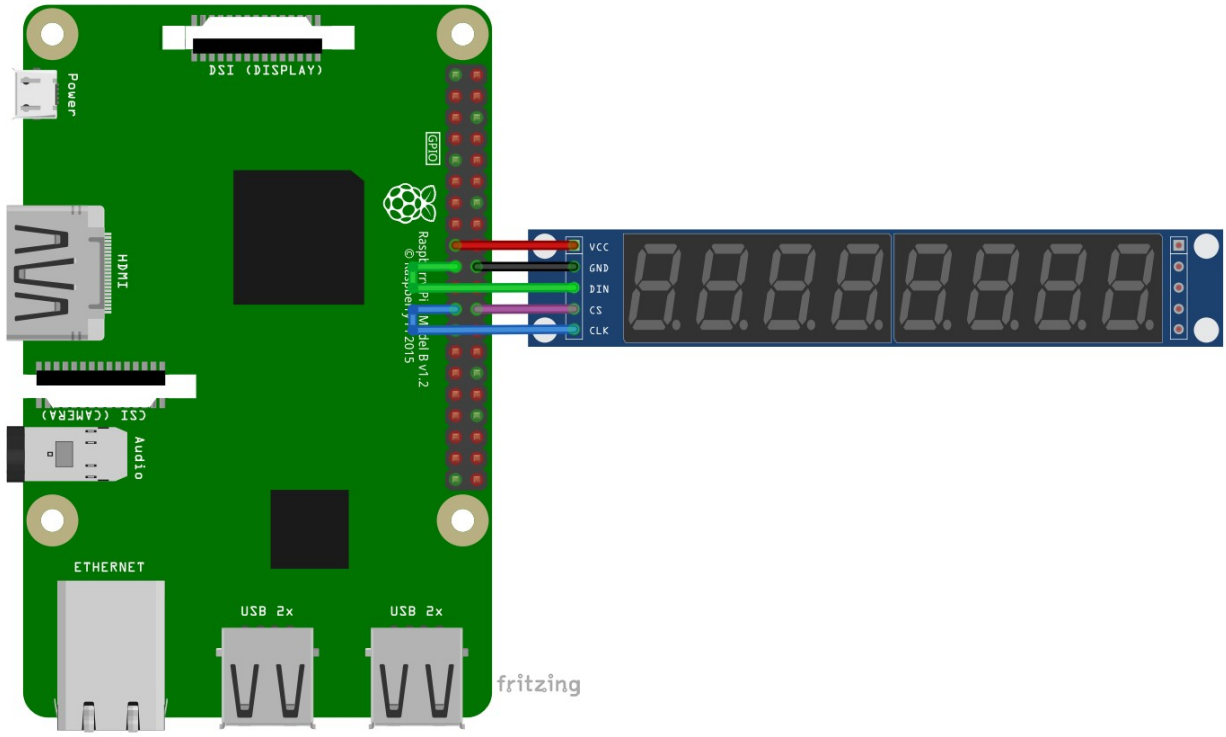
```
lc.setRow(0, number, 0x01)
```

Where hexadecimal value *0x01* is equal to *B00000001*, or turned *ON* the G segment of the seven segment screen.



Connecting the module with Raspberry Pi

Connect the module and the Raspberry Pi like on the connection diagram below:



Module pin	>	Raspberry Pi pin		
VCC	>	3V3	[pin 17]	**Red wire**
DIN	>	GPIO10	[pin 19]	**Green wire**
GND	>	GND	[pin 20]	**Black wire**
CLK	>	GPIO11	[pin 23]	**Blue wire**
CS	>	GPIO08	[pin 24]	**Purple wire**

Python library

In order to use the module with the Raspberry Pi it is recommended to download a library. We will use the library called "*luma.led_matrix*". Several steps need to be followed, in order to download and install this library. First is to update Raspbian, then to download and install the library. Open terminal and run the following commands to update the Raspbian:

```
sudo apt-get update
sudo apt-get upgrade -y
```

To download the library, run the following command:

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

After that, change directory to downloaded folder:

```
cd luma.led_matrix
```

To install the library run the following command:

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

Python script:

```
import time
from datetime import datetime
from luma.led_matrix.device import max7219
from luma.core.interface.serial import spi, noop
from luma.core.virtual import sevensegment

def date(seg):
    # Display current date on device
    now = datetime.now()
    seg.text = now.strftime('%d.%m.20%y')

def clock(seg, seconds): # Display current time on device
    for i in range(int(seconds / 0.5)):
        now = datetime.now()
        seg.text = now.strftime('%H-%M-%S')
        # calculate blinking dot
        if i % 2 == 0:
            seg.text = now.strftime('%H-%M-%S')
        else:
            seg.text = now.strftime('%H %M %S')

        time.sleep(0.5)

def show_message(seg, msg, delay=0.1): # Scrolls the message
    width = seg.device.width
    padding = ' ' * width
    msg = padding + msg + padding
    for i in range(len(msg)):
        seg.text = msg[i:i + width] # string slicing, msg[index:index]
        time.sleep(delay)
```

```
def main():
    serial = spi(port=0, device=0, gpio=noop())
    device = max7219(serial, cascaded=1)
    seg = sevensegment(device)

    print('Simple text...')
    seg.text = 'HELLO'
    time.sleep(2)
    seg.text = ' GOODBYE'
    time.sleep(2)

    print('Scrolling alphabet text...')
    show_message(seg, 'AZ-Delivery')
    show_message(seg, 'PI number 3.14159')
    s1 = '0123456789 '
    s2 = 'abcdefghijklmnopqrstuvwxyz '
    s3 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    show_message(seg, '{}{}{}'.format(s1, s2, s3))

    date(seg)
    time.sleep(5)
    clock(seg, seconds=10)

    print('Brightness...')
    for x in range(5):
        for intensity in range(16):
            seg.device.contrast(intensity * 16)
            time.sleep(0.1)
    device.contrast(0x7F)

try:
    while True:
        main()
        time.sleep(1)

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

Save this script under the name *"sevenSegments.py".* To run the script open terminal in the directory where you saved the script and run the following command: **`python3 Shock2.py`**

To stop the script, press *CTRL* + *C*.

The script starts with imports of libraries: time, datetime, and module related libraries.

Then we create four functions.

First function is called *`date()`* which is used to display current date. It accepts one argument and returns no value. The argument is called "*seg*" and it represents module object (later in text we will explain this object thoroughly). To get current date, use the following line of the code:

```
now = datetime.now()
```

The function returns unix timestamp, which is the number of seconds from *January the 1st 1970* up to current time of the Raspbian operating system.

To convert this timestamp into a readable string, use:

```
now.strftime('%d.%m.20%y')
```

where *%d* represents day, *%m* represents month and *%y* represents year. We use '*%d.%m.20%y*' which returns, for example, '*29.11.2019.*'. To output this date string value, we use *`seg.text`* property of the "*seg"* object.

The second function is called *clock()* which is used to display clock time. It accepts two arguments and returns no value. The first argument represents module object and second argument is number of seconds, a time interval after which clock time will be displayed on the module. Inside this function there is one *for* loop, which loops through the number of seconds in *seconds* argument.

In every loop the current time will be displayed in the format '*HH-MM-SS*' for half a second, and the other half second the same time will be displayed BUT in the format '*HH MM SS*'. This gives the effect of blinking dashes between time numbers.

The third function is called *show_message()* which is used to display scrolling text on the module. It accepts three arguments, where the last argument is optional. The first argument represents module object, the second argument is a string message that will be scrolled. The last, an optional argument is a scrolling interval shift, or we could say a scrolling speed. Default value of this argument is 0.1 seconds, and you can change it when you call the function on the following way:

```
show_message(seg, "message", delay=1.5)
```

Inside the *show_message()* function we loop through string message and slice it into smaller strings with length of module width (which is 8 characters, but if you are using multiple modules in Daisy chain, this will be higher value). Only sliced strings will be shown on the module, giving the effect of scrolling text.

The last function is called *main()* which is used to create all necessary objects and to run all other functions. It accepts no arguments and returns no values. At the beginning of this function we create three objects. The first object is for SPI interface, where we set the SPI hardware port *0*, the device number in a Daisy chain (in our case, this is *0*, because we have only one module), and GPIO pin names. The second object is used to set-up all the registers inside the driver chip MAX7219. We use SPI interface object to initialize driver chip object.

Also when we initialize driver chip object, we can specify how many modules are cascaded in the Daisy chain (in our case this value is *1*). And last object is the module object, called "*seg*", and to initialize it we use driver chip object. We have to create this object, because MAX7219 driver chip is used to drive other screen too. So with "*seg*" object we specify that we are driving seven segments with MAX7219 driver chip.

After this, we display some text and call other functions. First we display words '*HELLO*' and ' *GOODBYE*', then we scroll three messages. The first message is '*AZ-Delivery*', the second is '*PI number 3.14159*' and the last contains all numbers and all lowercase and uppercase letters. The last message is splitt into three strings because of better readability.

After that, we call the *date()* and the *clock()* functions. We set the *seconds* arguments of the *clock()* function to *10* seconds, so this function will output *10* seconds of current time.

Finally, at the end of *main()* function we change the brightness of the module. We do it with the following line of the code:

```
seg.device.contrast(number)
```

where the *number* is the level of brightness. There are 16 different brightness level values, but this value has to be a number which when divided with 16 returns integer number, because of MAX7219 driver chip internal registers are set in this manner.

At the end of the script, we create *try-except* block with infinite loop block (*while True:*) inside. In the infinite loop block, we call the *main()* function.

To stop the script, press *CTRL + C*. This is called the keyboard interrupt, and we wait for it in the except block:

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

You've done it!

Now you can use your module for various projects.

Now is the time to learn and make the Projects on your own. You can do that with the help of many example scripts and other tutorials, which you can find 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