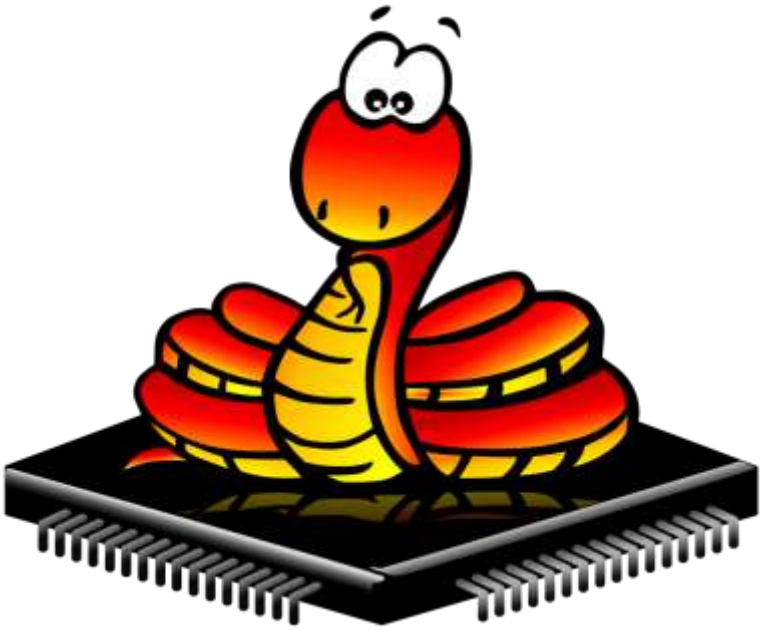
EasyPython.py



Library Creator

Md Hasemi Rafsan Jani Shohan

BSC in EEE

Daffodil International University

Embedded System Engineer

```
from EasyPython import *
from EasyPython2 import *

while True:
    #blynk(pin,time_on,time_off)
    #digitalWrite(pin,Logic)
    #digitalRead(pin)
    #delay(ms)
    #analogRead()
    #analogWrite(pin,value)
    #looping(m,n)
    #lcd(message)
    #clear()
    #map(x,min1,max1,min2,max2)
    #constrain(x,mi,mx)
    #increasing(c)
    #decreasing(c)
    #Serial_write(x)
```

```
class RTC():
        from machine import RTC
        rtc = RTC()
        rtc.datetime((2017, 8, 23, 1, 12, 48, 0, 0)) # set a specific date and time

try:

        def add(a,b):
                return a+b

        def sub(a,b):
                return a-b


        def blynk(pin,time_on,time_off):
                from machine import Pin
                import time
                p0 = Pin(pin,Pin.OUT)
                p0.on()
                time.sleep(time_on)
                p0.off()
                time.sleep(time_off)

        def digitalWrite(pin,Logic):
                        from machine import Pin
                        p0 = Pin(pin,Pin.OUT)
```

```
            p0.value(Logic)

def digitalRead(pin):
    from machine import Pin
    p0 = Pin(pin,Pin.IN)
    return p0.value()

def delay(ms):
    import time
    time.sleep(ms/1000)

def analogRead():
    from machine import ADC
    adc = ADC(0) # create ADC object on ADC pin
    v = adc.read() # read value, 0-1024
    return v
# unc
def analogWrite(pin,value):
    from machine import Pin, PWM
    value = int(map(value,0,255,0,1023))
    pwm2 = PWM(Pin(pin), freq=500, duty=value)

#unc

#test
def looping(a,b,c,d,e,m,n,time):
    from machine import Pin
    import time
```

```
        for i in range(m,n):
            listt = [a,b,c,d,e] #pin number
            cc = listt[i]
            Pin(cc,Pin.OUT).on()
            time.sleep(int(time/1000))
            Pin(cc,Pin.OUT).off()

    def lcd(message):
        import machine
        from machine import Pin,I2C
        from lcd_api import LcdApi
        from i2c_lcd import I2cLcd
        from time import sleep
        I2C_ADDR = 0x27
        totalRows = 2
        totalColumns = 16
        i2c = I2C(scl=Pin(5), sda=Pin(4), freq=10000)      #initializing the I2C method for
ESP8266
        lcd = I2cLcd(i2c, I2C_ADDR, totalRows, totalColumns)
        lcd.putstr(f"{message}")

    def clear():
        import machine
        from machine import Pin,I2C
        from lcd_api import LcdApi
        from i2c_lcd import I2cLcd
        from time import sleep
        I2C_ADDR = 0x27
        totalRows = 2
```

```
        totalColumns = 16
        i2c = I2C(scl=Pin(5), sda=Pin(4), freq=10000)      #initializing the I2C method for
ESP8266
        lcd = I2cLcd(i2c, I2C_ADDR, totalRows, totalColumns)
        lcd.clear()


    def map(x,min1,max1,min2,max2):
        x = constrain(x,min1,max1)
        try:
            c = (x/max1)*max2
            return c
        except:
            c = 0



    def constrain(x,mi,mx):
        if x < mi :
            return mi
        elif x > mx:
            return mx
        else :
            return x
    #formate
    #c = f
    #f = decreasing(c)
```

```
    def increasing(c,step):
        c = c + step
        return c

    def decreasing(c,step):
        c = c - step
        return c

    def Serial_write(x):
        from machine import UART
        uart = UART(0, baudrate=115200)
        uart.write(f'{x}\n')

except:
    pass
```

Code 1

```
#from EasyPython2 import *
class lcdbegin():
    import machine
    from machine import Pin,I2C
    from lcd_api import LcdApi
```

```
    from i2c_lcd import I2cLcd
    I2C_ADDR = 0x27
    totalRows = 2
    totalColumns = 16
    i2c = I2C(scl=Pin(5), sda=Pin(4), freq=10000)
    lcd = I2cLcd(i2c, I2C_ADDR, totalRows, totalColumns)
```

Code2

```
#from EasyPython2 import *
class lcdbegin():
    import machine
    from machine import Pin,I2C
    from lcd_api import LcdApi
    from i2c_lcd import I2cLcd
    I2C_ADDR = 0x27
    totalRows = 2
    totalColumns = 16
    i2c = I2C(scl=Pin(5), sda=Pin(4), freq=10000)
    lcd = I2cLcd(i2c, I2C_ADDR, totalRows, totalColumns)
```

code3

```
"""Provides an API for talking to HD44780 compatible character LCDs."""

import time

class LcdApi:
    """Implements the API for talking with HD44780 compatible character LCDs.
    This class only knows what commands to send to the LCD, and not how to get
    them to the LCD.
```

```
It is expected that a derived class will implement the hal_xxx functions.
"""

# The following constant names were lifted from the avrlib lcd.h
# header file, however, I changed the definitions from bit numbers
# to bit masks.
#
# HD44780 LCD controller command set

LCD_CLR = 0x01              # DB0: clear display
LCD_HOME = 0x02             # DB1: return to home position

LCD_ENTRY_MODE = 0x04       # DB2: set entry mode
LCD_ENTRY_INC = 0x02        # --DB1: increment
LCD_ENTRY_SHIFT = 0x01      # --DB0: shift

LCD_ON_CTRL = 0x08          # DB3: turn lcd/cursor on
LCD_ON_DISPLAY = 0x04       # --DB2: turn display on
LCD_ON_CURSOR = 0x02        # --DB1: turn cursor on
LCD_ON_BLINK = 0x01         # --DB0: blinking cursor

LCD_MOVE = 0x10             # DB4: move cursor/display
LCD_MOVE_DISP = 0x08        # --DB3: move display (0-> move cursor)
LCD_MOVE_RIGHT = 0x04       # --DB2: move right (0-> left)

LCD_FUNCTION = 0x20         # DB5: function set
LCD_FUNCTION_8BIT = 0x10    # --DB4: set 8BIT mode (0->4BIT mode)
```

```
LCD_FUNCTION_2LINES = 0x08  # --DB3: two lines (0->one line)
LCD_FUNCTION_10DOTS = 0x04  # --DB2: 5x10 font (0->5x7 font)
LCD_FUNCTION_RESET = 0x30   # See "Initializing by Instruction" section

LCD_CGRAM = 0x40           # DB6: set CG RAM address
LCD_DDRAM = 0x80           # DB7: set DD RAM address

LCD_RS_CMD = 0
LCD_RS_DATA = 1

LCD_RW_WRITE = 0
LCD_RW_READ = 1

def __init__(self, num_lines, num_columns):
    self.num_lines = num_lines
    if self.num_lines > 4:
        self.num_lines = 4
    self.num_columns = num_columns
    if self.num_columns > 40:
        self.num_columns = 40
    self.cursor_x = 0
    self.cursor_y = 0
    self.implied_newline = False
    self.backlight = True
    self.display_off()
    self.backlight_on()
    self.clear()
    self.hal_write_command(self.LCD_ENTRY_MODE | self.LCD_ENTRY_INC)
```

```
    self.hide_cursor()
    self.display_on()

def clear(self):
    """Clears the LCD display and moves the cursor to the top left
    corner.
    """
    self.hal_write_command(self.LCD_CLR)
    self.hal_write_command(self.LCD_HOME)
    self.cursor_x = 0
    self.cursor_y = 0

def show_cursor(self):
    """Causes the cursor to be made visible."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                           self.LCD_ON_CURSOR)

def hide_cursor(self):
    """Causes the cursor to be hidden."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

def blink_cursor_on(self):
    """Turns on the cursor, and makes it blink."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                           self.LCD_ON_CURSOR | self.LCD_ON_BLINK)

def blink_cursor_off(self):
    """Turns on the cursor, and makes it no blink (i.e. be solid)."""
```

```
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                           self.LCD_ON_CURSOR)

def display_on(self):
    """Turns on (i.e. unblanks) the LCD."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

def display_off(self):
    """Turns off (i.e. blanks) the LCD."""
    self.hal_write_command(self.LCD_ON_CTRL)

def backlight_on(self):
    """Turns the backlight on.

    This isn't really an LCD command, but some modules have backlight
    controls, so this allows the hal to pass through the command.
    """
    self.backlight = True
    self.hal_backlight_on()

def backlight_off(self):
    """Turns the backlight off.

    This isn't really an LCD command, but some modules have backlight
    controls, so this allows the hal to pass through the command.
    """
    self.backlight = False
    self.hal_backlight_off()
```

```
def move_to(self, cursor_x, cursor_y):
    """Moves the cursor position to the indicated position. The cursor
    position is zero based (i.e. cursor_x == 0 indicates first column).
    """
    self.cursor_x = cursor_x
    self.cursor_y = cursor_y
    addr = cursor_x & 0x3f
    if cursor_y & 1:
        addr += 0x40    # Lines 1 & 3 add 0x40
    if cursor_y & 2:    # Lines 2 & 3 add number of columns
        addr += self.num_columns
    self.hal_write_command(self.LCD_DDRAM | addr)

def putchar(self, char):
    """Writes the indicated character to the LCD at the current cursor
    position, and advances the cursor by one position.
    """
    if char == '\n':
        if self.implied_newline:
            # self.implied_newline means we advanced due to a wraparound,
            # so if we get a newline right after that we ignore it.
            pass
        else:
            self.cursor_x = self.num_columns
    else:
        self.hal_write_data(ord(char))
        self.cursor_x += 1
```

```
    if self.cursor_x >= self.num_columns:
        self.cursor_x = 0
        self.cursor_y += 1
        self.implied_newline = (char != '\n')
    if self.cursor_y >= self.num_lines:
        self.cursor_y = 0
    self.move_to(self.cursor_x, self.cursor_y)

def putstr(self, string):
    """Write the indicated string to the LCD at the current cursor
    position and advances the cursor position appropriately.
    """
    for char in string:
        self.putchar(char)

def custom_char(self, location, charmap):
    """Write a character to one of the 8 CGRAM locations, available
    as chr(0) through chr(7).
    """
    location &= 0x7
    self.hal_write_command(self.LCD_CGRAM | (location << 3))
    self.hal_sleep_us(40)
    for i in range(8):
        self.hal_write_data(charmap[i])
        self.hal_sleep_us(40)
    self.move_to(self.cursor_x, self.cursor_y)

def hal_backlight_on(self):
```

```
    """Allows the hal layer to turn the backlight on.

    If desired, a derived HAL class will implement this function.
    """
    pass

def hal_backlight_off(self):
    """Allows the hal layer to turn the backlight off.

    If desired, a derived HAL class will implement this function.
    """
    pass

def hal_write_command(self, cmd):
    """Write a command to the LCD.

    It is expected that a derived HAL class will implement this
    function.
    """
    raise NotImplementedError

def hal_write_data(self, data):
    """Write data to the LCD.

    It is expected that a derived HAL class will implement this
    function.
    """
    raise NotImplementedError
```

```
    def hal_sleep_us(self, usecs):
        """Sleep for some time (given in microseconds)."""
        time.sleep_us(usecs)
```

Code4

```
try:
        a = ''
        b = ''

        def begin(a,b):
                import ntptime
                import network
                #import time
                from machine import RTC #
                import time #sleep
                timeout =0
                wifi = network.WLAN(network.STA_IF)

                # Restarting WiFi
                wifi.active(False)
                time.sleep(0.5)
                wifi.active(True)

                wifi.connect(a,b)

                if not wifi.isconnected():
                        print('connecting..')
                        while (not wifi.isconnected() and timeout < 5):
```

```
                print(5 - timeout)
                timeout = timeout + 1
                time.sleep(1)

        if(wifi.isconnected()):
                print('Connected')
        else:
                print('Time Out')



        #(2000, 1, 1, 0, 40, 8, 5, 1)


def operate(utf):
        utf = int(utf)
        from machine import RTC #
        import time #sleep
        import ntptime
        import network
        from machine import RTC
        rtc = RTC()
        ntptime.settime()
        UTC_OFFSET = +utf * 60 * 60 #>>for bd +6
        actual_time = time.localtime(time.time() + UTC_OFFSET)
        c = actual_time
        return c
```

```
except:
    pass
```

Code 5