Documentation micro:bit

Module import: from micro:bit import *

Global function calls:

Function	Action
panic(n)	blocks the system and displays infinitely a "sad face" image followed by n (for developers only)
reset()	restarts the board (and executes main.py)
sleep(dt)	stops program for dt milliseconds
running_time()	returns the time in milliseconds since the board was switched on or resetted
temperature()	returns the temperature in degrees celsius (as float)

Class MicroBitButton

button_a	object reference (instance) of button A
button_b	object reference (instance) of button B

Methods:

is_pressed()	returns True, if the button is down (pressed); otherwise False
was_pressed()	returns True, if the button was pressed since the last call (or start of the program). Another call returns False, until the button is pressed again
get_presses()	returns the number of button presses since the last call (or start of program). Another call returns False, until the button is pressed again

Example:

```
if button_a.was_pressed():
    do_something
```

Class MicroBitDisplay

display	object reference (instance)

Methods:

set_pixel(x, y, value)	sets the intensity of pixel at position x, y. value in range 0..9
clear()	clears all pixels
show(str)	shows given str on LED display. If str has more than 1 character, the text ist scrolled until the last character becomes visible
show(list_of_img, delay = 400, loop = False, wait = True, clear = False)	shows all images of the list in a timed sequence. If loop = True ist, the sequence is repeated infinitely. For wati = True the method is blocking, otherwise it returns while the sequence continues in the background. delay is the display time for each image in ms (default: 400). For clear = True, the display is clear after the last image is shown
show(img)	show Image img on LED display. If img is bigger than 5x5 pixel, the pixels in range x, y = 0..4 are shown. If img is smaller than 5x5 pixel, the missing pixels are black
scroll(str)	shows str as scrolling text. The last character disappears
off()	turns off display (pin3, pin4, pin6, pin7, pin9, pin19 may be used for general digital in/out)
on()	turns on the display and sets pin3, pin4, pin6, pin7, pin9, pin19 in display mode
scroll(str, delay = 150, loop = False, wait = True, monospace = False)	show str as scrolling text . If loop = True, the sequence is repeated infinitely. For wait = True the method is blocking, otherwise it returns while the sequence continues in the background. delay is the display time for a single column (default: 150)

Examples:

```
display.show("A")
display.scroll("Hello")
display.show([Image.HAPPY, Image.SAD])
```

Class MicroBitImage

Image(str)	creates object reference (instance). str has format "aaaaa:bbbbb:ccccc:ddddd:eeeee:" where a is a number in the range 0..9 and defines the intensity of the pixel, a are values of first line, b of second, etc.
Image()	creates object reference (instance) with 5x5 pixels and all pixels are turned off
Image(width, height)	creates object reference (instance) with given number of horizontal and vertical pixels with all pixels turned off (value = 0)

Methods:

set_pixel(x, y, value)	sets the intensity of pixel at position x, y. value in range 0..9
shift_left(n)	returns an Image object by shifting the picture left by n columns
shift_right(n)	returns an Image object by shifting the picture right by n columns
shift_up(n)	returns an Image object by shifting the picture up by n rows
shift_down(n)	returns an Image object by shifting the picture down by n rows
copy()	returns a clone of the image
invert()	returns an Image object by inverting the intensity of all pixels (new_value = 9 - value)
fill(value)	returns an Image with all pixel intensities set to to given value (0..9)
dest.blit(img, x, y, w, h, xdest, ydest)	copies from given image img a rectangular section of size w, h at position x, y to dest image at position xdest, ydest

Operations:

image_new = image * n	returns an object with all pixel intensities multiplied by n
image_new = image1 + image2	returns an object by adding the pixel intensities of image1 and image2

Predefined Images:

- Image.HEART
- Image.HEART_SMALL
- Image.HAPPY
- Image.SMILE
- Image.SAD
- Image.CONFUSED
- Image.ANGRY
- Image.ASLEEP
- Image.HOUSE
- Image.TORTOISE
- Image.BUTTERFLY
- Image.SURPRISED
- Image.SILLY
- Image.FABULOUS
- Image.MEH
- Image.YES
- Image.NO
- Image.RABBIT
- Image.COW
- Image.TSHIRT
- Image.ROLLERSKATE
- Image.DUCK
- Image.TRIANGLE
- Image.TRIANGLE_LEFT
- Image.CHESSBOARD
- Image.DIAMOND
- Image.DIAMOND_SMALL
- Image.SQUARE
- Image.SQUARE_SMALL
- Image.STICKFIGURE
- Image.GHOST
- Image.SWORD
- Image.GIRAFFE
- Image.MUSIC_CROTCHET
- Image.MUSIC_QUAVER
- Image.MUSIC_QUAVERS
- Image.PITCHFORK
- Image.XMAS
- Image.PACMAN
- Image.TARGET
- Image.UMBRELLA
- Image.SNAKE
- Image.SKULL
- Image.CLOCK12, Image.CLOCK11, Image.CLOCK10, Image.CLOCK9, Image.CLOCK8, Image.CLOCK7, Image.CLOCK6, Image.CLOCK5, Image.CLOCK4, Image.CLOCK3, Image.CLOCK2, Image.CLOCK1
- Image.ARROW_N, Image.ARROW_NE, Image.ARROW_E, Image.ARROW_SE, Image.ARROW_S, Image.ARROW_SW, Image.ARROW_W, Image.ARROW_NW
- Lists:Image.ALL_CLOCKS, Image.ALL_ARROWS

Remark:
A MicroBitImage object (short "image") is an abstraction of the real picture and gets visible when display.show(img) is called.

An image may have any number of horizontal and vertical pixels (w, h), but only pixels in the range x = 0..4, y = 0..4 are shown. (If image is smaller, the non-defined pixels are black.) For bigger images, use blit()to extract a section.

Be aware that with the exception of set_pixel() and blit() the methods do not modify the image itself, but return a new image. To change img, it must be assigned to the return value.

Example:
```
img = Image(2, 2)
img = img.invert()
display.show(img)
```

Class MicroBitTouchPin

pin0, pin1, pin2, pin8, pin12, pin16	instances for general digital-in/digital-out
pin0, pin1, pin2	instances for analog-in/analog-out (PWM)
pin3, pin4, pin6, pin7, pin9, pin10	instances predefined for LED display (display mode)
pin5, pin11	instances predefine for button A, B (button mode)
pin13, pin14, pin15	instances predefined for SPI (spi mode)
pin19, pin20	instances predefined for I2C (i2c mode)

Methods:

read_digital()	returns True, if the pin is logical 1 (HIGH); returns False, if logical 0 (LOW) (Pulldown 10 kOhm)
write_digital(v)	if v = 1, sets the pin to logical 1 (HIGH); if v = 0, sets the pin to logical 0 (LOW) (max. current: 5 mA)
read_analog()	returns value from ADC in range 0..1023 (input impedance: 10 MOhm)
write_analog(v)	sets the PWM duty cycle (v = 0..1023 corresponds to 0..100%) (max. current: 5 mA)
set_analog_period(period)	sets the PWM period in milliseconds
set_analog_period_microseconds(period)	sets the PWM period in microseconds (> 300)

Class MicroBitAccelerometer

accelerometer	object reference (instance)

Methods:

get_x(), get_y(), get_z()	returns the current value of acceleration in x, y or z direction (int, range approx.. -2047 to +2048, corresponding approx. -20 m/s^2 to +20 m/s^2, gravitational acceleration of approx. 10 m/s^2 included). x direction: ButtonA-ButtonB; y direction: Pin2-USB; z direction: perpenticular to board
get_values()	returns a tuple with the acceleration in x, y and z direction (ints, units as above)

Class MicroBitCompass

compass	object reference (instance)

Methods:

calibrate()	starts a blocking calibration routine needed to get accurate readings: tilt the micro:bit in different directions, so that the blinking pixel reaches the border, where the border pixel is turned on. When all border pixels are lit, the program continues
is_calibrated()	returns True, if the compass was calibrated
clear_calibration()	resets the compass to the non-calibrated state
heading()	returns the angle of the current direction if the micro:bit with respect to the north
get_x(), get_y(), get_z()	returns the current value of the x, y or z component of the magnetic field at the sensor position (int, in microtesla, no calibration needed)
get_values()	returns a tuple with x , y and z components of the magnetic field at the sensor position (int, in microtesla, no calibration needed)

Class NeoPixel
(**Module import: from neopixel import ***)

np = NeoPixel(pin, n)	creates a Neopixel object (instance) with n neopixels controlled via given pin. Each pixel is addressed by its position (starting from 0) and its color is determined by assigning a RGB tuple, e.g. np[2] = (0, 100, 255) sets pixel # 2 to red = 0, green = 100, blue = 255. *show()* must be called to update the display (Strips with WS2812 LEDs supported.)

Methods:

clear()	clears all pixels
show()	shows the pixels. Must be called to make any change of color values visible

Module music
(**Module import: from music import ***) **Functions:**

set_tempo(bpm = 120)	sets the number of beats per minute (default: 120)
pitch(frequency, len, pin = microbit.pin0, wait = True)	plays a tone with given frequency in Hertz during the given length (duration) (in milliseconds). pin defines the output pin at the GPIIO header (default: P0). If wait = True, the function is blocking; otherwise it returns while the sound continues (until finished or stop() is called)
play(melody, pin = microbit.pin0, wait = True, loop = False)	plays a melody with current tempo. pin defines the output pin at the GPIIO header (default: P0). If wait = True, the function is blocking; otherwise it returns while the sound continues (until finished or stop() is called). If loop is True, the melody is played again infinitely
stop(pin = microbit.pin0)	stops sound output at given header pin (default: P0)

Remark:
A melody is a list of strings in the format ["note:duration", "note:duration",...]
note in musical notation: c, d, e, f, g, a, h with optional octave number (default: 1): e.g. c2, d2, ... and optional sharp: c#, d#,... or c#2, d#2,...
duration in number of ticks (optional, defaut: 1)

Predifined song lists:

- ADADADUM - the opening to Beethoven's 5th Symphony in C minor
- ENTERTAINER - the opening fragment of Scott Joplin's Ragtime classic The Entertainer
- PRELUDE - the opening of the first Prelude in C Major of J.S.Bach's 48 Preludes and Fugues
- ODE - the Ode to Joy theme from Beethoven's 9th Symphony in D minor
- NYAN - the Nyan Cat theme
- RINGTONE - something that sounds like a mobile phone ringtone. To be used to indicate an incoming message
- FUNK - a funky bass line for secret agents and criminal masterminds
- BLUES - a boogie-woogie 12-bar blues walking bass
- BIRTHDAY - Happy Birthday to You...
- WEDDING - the bridal chorus from Wagner's opera Lohengrin
- FUNERAL - the funeral march otherwise known as Frédéric Chopin's Piano Sonata No. 2 in B×7; minor.
- PUNCHLINE - a fun fragment that signifies a joke has been made
- PYTHON - John Philip Sousa's march Liberty Bell aka, the theme for Monty Python's Flying Circus
- BADDY - silent movie era entrance of a baddy
- CHASE - silent movie era chase scene
- BA_DING - a short signal to indicate something has happened
- WAWAWAWAA - a very sad trombone
- JUMP_UP - for use in a game, indicating upward movement
- JUMP_DOWN - for use in a game, indicating downward movement
- POWER_UP - a fanfare to indicate an achievement unlocked
- POWER_DOWN - a sad fanfare to indicate an achievement lost

Module radio:
(**Module import: from radio import ***)
Computer communication over Bluetooth Functions:

on()	turns Bluetooth communication on. Connecting to a micro:bit node that has radio turned on
off()	turns Bluetooth communicaion off
send(msg)	sends the message string to the receiving node's message queue (First-In-First-Out, FIFO buffer)
msg = receive()	returns the oldest message (string) in the message queue and removes it from the queue. None is returned, if the queue is empty. It is assumed that the messages is sent with send(smg), so it can be converted in a valuable string [otherwise a ValueError exception ("received packet is not a string") is raised]
send_bytes(msg_bytes)	sends the message bytes (class *bytes, e.g b'\x01*\x48') to the receiving node's message queue (First-In-First-Out, FIFO buffer)
receive_bytes()	returns the oldest message bytes from the message queue and deletes it from the queue. None is returned, if the queue is empty. To send the message, send_bytes(msg) must be used (and not send(msg))

Module mbglow:
Modul import: from mbglow import *

Functions:

makeGlow()	creates a visible glowbug at position (0, 0) directed to the north with its trace enabled. Coordinate system: -2 <=x <=2 (+ to the right), -2 <= y <= 2 (+ upwards), (0, 0) at center pixel
setSpeed()	sets the speed for movements (0..100)
show()	enables the visibility for the following movements
hide()	disables the visibility
clear()	clears all visible pixels. The glowbug remains at the current position (but is invisible)
showTrace(enable)	enables/disables the trace for the following movements (pixels are turned on at the glowbug position)
forward()	moves the glowbug one step in the forward direction
back()	moves the glowbug one step in the backward direction
left(angle)	turns the glowbug in 45 degrees increments to the left (angle = 45, 90, 135, 180, 215, 270, 315)
right(angle)	urns the glowbug in 45 degrees increments to the right (angle = 45, 90, 135, 180, 215, 270, 315)
setPos(x, y)	sets the glowbug at position (x, y)
getPos()	returns the current position of the glowbug (as tuple)
isLit()	returns True, if the pixel at the current glowbug position is turned on

Module maqueen
(Mini:Maqueen Rover, Realmode)

(Module import: from maqueen import *)

setSpeed()	sets the speed for all movements (0..100)
forward()	sets the Maqueen in forward motion
backward()	sets the Maqueen in reverse motion
left()	sets the Maqueen in a left turn (one motor in forward, the other in backward rotation)
right()	sets the Maqueen in a right turn (one motor in forward, the other in backward rotation)
leftArc(radius)	sets the Maqueen to a left curve with given radius (in cm, approx.)
rightArc(radius)	sets the Maqueen to a right curve with given radius (in cm, approx.)
stop()	stops the movement
getDistance()	returns the distance (in mm) measured with the ultrasonic sensor

Module linkup

(ESP32 Coprocessor Board connected to I2C)

(Module import: from linkup import *)

connectAP(ssid, password)	connects the LinkUp with an existing access point (hotspot, router) with ssid and password
httpGet(url)	executes an HTTP GET request and returns the response. url in the form "http://<server>?key=value&key=value&...". https can also be used instead of http
httpPost(url, content)	executes an HTTP POST request and returns the response. url in the form "http://<server>". content in the format "key=value&key=value&...". https can also be used instead of http
httpDelete(url)	executes an HTTP DELETE request with the given resource (file name)
startHTTPServer(ssid, password, startAP = True, reboot = False)	starts an HTTP server (web server on port 80). For startAP = True a local access point with given ssid and password is started. If startAP = False, an existing access point with ssid and password is used. If reboot = True, the LinkIn is rebooted first (to stop a running server)
getRequestParams()	returns the GET request parameters as a dictionary {key : value}
sendReply(reply)	sends a list of string replies to the LinkUp that are included in the HTML response

Module sht31

(Sensirion temperature and air humidity sensor connected to I2C)

(Module import: from sht31 import SHT31)

sht = SHT31	creates a sensor instance
sht.getTempHumi()	returns a tuple with temperature (in degrees Celsius) and humidity (in percent)