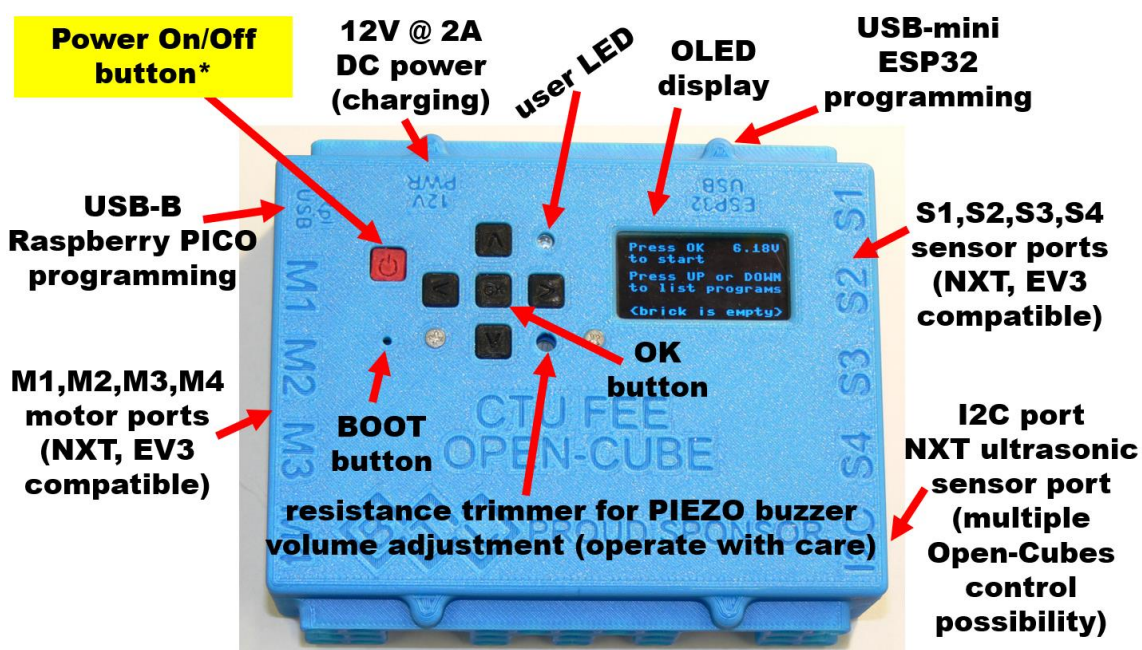


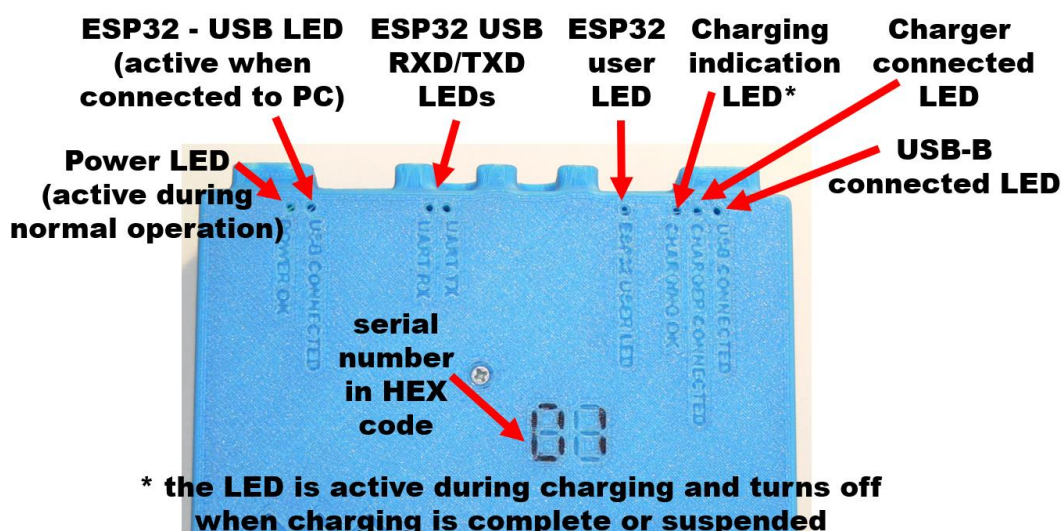
# OPEN-CUBE QUICK START GUIDE

For more information please visit: [open-cube.fel.cvut.cz](http://open-cube.fel.cvut.cz)

1. Switch ON the Open-Cube and connect it to the computer via USB-B (look for COMport number in the device manager)
2. Use Thonny software to create and debug Micropython programs (select Raspberry Pi PICO as a platform and the right COMport, go to Tools/Options/Interpreter...)
3. See the next page for a list of most common commands, check the full description of available functions on following pages for more detailed information and useful tips.



**\*a long press (>5s) of the Power button turns the Open-Cube OFF (secured by hardware circuit, will work even if firmware is corrupted)**



# LIST OF MOST USEFUL MICROPYTHON O-C COMMANDS

## TIMING

```
from time import sleep , sleep_ms , sleep_us
sleep(1)          # Sleep program for 1 second
sleep_ms(2)      # Sleep program for 2 miliseconds
sleep_us(3)      # Sleep program for 3 microseconds look also for "Timer"
```

## MOTORS (the same for NXT and EV3), more in chapter 7

```
from lib. robot_consts import Port      # import constants
robot.init_motor(Port.M1)              # initialize the motor
robot.init_motor(Port.M2)              # initialize the motor
robot.motors[Port.M1].set_power(50)    # set power to 50%, clock-wise
robot.motors[Port.M2].set_power(-20)   # set power to 20%, counter-clock-wise
```

## SENSORS

### NXT TOUCH

```
from lib. robot_consts import Sensor, Port      # import constants, sensor types...
robot.init_sensor(sensor_type = Sensor.NXT_TOUCH, port = Port.S1) # init the sensor on port S1
touch_pressed = robot.sensors.touch[Port.S1].pressed() #read the state of the button
```

### NXT OPTICAL see chapter 4.2 for more functions!

```
from lib. robot_consts import Sensor, Port      # import constants, sensor types...
robot.init_sensor(sensor_type = Sensor.LIGHT, port = Port.S1) # init optical sensor at port S1
robot.sensors.light[Port.S1].on()              #switch on the LED
light_intensity = robot.sensors.light[Port.S1].intensity() # measure the light intensity
```

### NXT MICROPHONE

```
from lib. robot_consts import Sensor, Port      # import constants, sensor types...
robot.init_sensor(sensor_type = Sensor.NXT_SOUND, port = Port.S1) # init the sensor on port S1
sound_intensity = robot.sensors.sound[Port.S1].intensity() # read the sound intensity
```

### NXT ULTRASONIC

```
from lib. robot_consts import Sensor      # import constants, sensor types...
robot.init_sensor(sensor_type = Sensor.NXT_ULTRASONIC) # initialize the sensor
distance = robot.sensors.ultra_nxt.distance() # measure distance
```

### NXT GYROSCOPE - currently not implemented (I2C interface, manufactured by HiTechnic)

### EV3 TOUCH

```
from lib. robot_consts import Sensor, Port      # import constants, sensor types...
robot.init_sensor(sensor_type = Sensor.EV3_TOUCH, port = Port.S1) # initialize the sensor
touch_pressed = robot.sensors.touch[Port.S1].pressed() #read the state of the button
```

### EV3 OPTICAL see chapter 5.2 for more functions (color sensing for example)

```
from lib. robot_consts import Sensor, Port      # import constants, sensor types...
robot.init_sensor(sensor_type = Sensor.EV3_COLOR, port = Port.S1) # initialize the sensor
reflection = robot.sensors.light[Port.S1].reflection() # measure the light intensity
```

### EV3 ULTRASONIC

```
from lib. robot_consts import Sensor, Port      # import constants, sensor types...
robot.init_sensor(sensor_type = Sensor.EV3_ULTRA, port = Port.S1) # initialize the sensor
distance = robot.sensors.ultrasonic[Port.S1].distance() # measure the distance
```

### **EV3 GYROSCOPE** (single axis)

```
from lib.robot__consts import Sensor, Port          # import constants, sensor types...
robot.init__sensor(sensor__type = Sensor.EV3__GYRO, port = Port.S1)      # initialize the sensor
robot.sensors.gyro[Port.S1].reset__angle(0)        # reset angle
(angle, speed) = robot.sensors.gyro[Port.S1].angle__and__speed() # measure the angular velocity
in one axis and position
```

### **O-C OPTICAL** see chapter 6.1 for other modes (color sensing, blue and green light - make POLICE car beacon effect...)

```
from lib.robot__consts import Sensor, Port          # import constants, sensor types...
robot.init__sensor(sensor__type = Sensor.OC__COLOR, port = Port.S1)      # initialize the sensor
reflection = robot.sensors.light[Port.S1].reflection() # measure the reflected red light intensity
```

### **O-C LASER** see chapter 6.2

```
from lib.robot__consts import Sensor, Port          # import constants, sensor types...
robot.init__sensor(sensor__type = Sensor.OC__LASER, port = Port.S1)      # initialize the sensor
distance = robot.sensors.laser[Port.S1].distance() # measure the distance
```

### **INTERNAL GYROSCOPE** see chapter 3.6 for example of sensor readout using interrupts

```
robot.init__sensor(sensor__type = Sensor.GYRO__ACC)
a__g__data = (0, 0, 0, 0, 0, 0) # ax:-, ay:-, az:-, gx:°/s, gy:°/s, gz:°/s
a__g__data = robot.sensors.gyro__acc.read__value() # measure the accelerations and angular rates
```

### **MISCELLANEOUS**

#### **DISPLAY** see chapter 3.5 for complete list of functions

```
robot.display.fill(0) # clear display
robot.display.text("test message",0,0,1) # print text
x, y = 1.23, 56.789 # set some variables
robot.display.text(f"x={x:.2 f}, y: {y:.2f}", 0, 10, 1) # print two float numbers
robot.display.show() # show the framebuffer
```

#### **SOUND - PIEZO**

```
robot.buzzer.set__freq__duty(4000,50) # 4000 Hz, 50 % duty
robot.buzzer.off() # stop the PWM generation
```

#### **BUTTONS**

```
from lib. robot__consts import Button # import constants, sensor types...
buttons = robot.buttons.pressed() # read the state
if buttons[Button.LEFT]: break # use it in an if condition
# other buttons: Button.LEFT, Button.RIGHT, Button.OK, Button.UP, Button.DOWN
```

#### **LED** (red, on the Open-Cube front panel)

```
robot.led.on() # switch LED on
robot.led.off() # switch LED off
```

#### **BATTERY**

```
voltage = robot.battery.voltage() #read the internal O-C Li-ion battery voltage
```

#### **I2C MASTER, I2C SLAVE** (Open-Cube control, or general usage), currently not implemented

#### **BLUETOOTH** serial communication & **Wi-Fi** access point – see chapter 3.7.2

```
MicroPython also supports: if x < 9: while x < 9: for y in range(0, 9):
print("debug message!") # print debug message into Thonny console
def add(number1, number2): return number1 + number2 # function definition
add(1, 2) # function usage
```

# Open-Cube

Katedra měření ČVUT FEL

2023

## Obsah

<b>1</b>	<b>Instalace firmwaru</b>	<b>4</b>
<b>2</b>	<b>Editor kódu</b>	<b>4</b>
2.1	Nastavení editoru . . . . .	4
2.2	Nahrání kódu do kostky . . . . .	4
<b>3</b>	<b>Robot</b>	<b>5</b>
3.1	Tlačítka . . . . .	7
3.2	LED . . . . .	8
3.3	Bzučák . . . . .	8
3.4	Baterie . . . . .	9
3.5	Displej . . . . .	10
3.6	Gyroskop a akcelerometr . . . . .	13
3.7	ESP32 komunikace . . . . .	15
3.7.1	Bluetooth propojení ESP32 s jiným zařízením . . . . .	15
3.7.2	Kostka . . . . .	17
3.7.3	Serial Bluetooth Terminal . . . . .	20
3.7.4	Matlab . . . . .	20
3.7.5	Simulink . . . . .	21
<b>4</b>	<b>NXT senzory</b>	<b>25</b>
4.1	Tlačítko . . . . .	25
4.2	Světelný senzor . . . . .	25
4.3	Ultrazvukový dálkoměr . . . . .	27
4.4	Zvukový senzor . . . . .	28
<b>5</b>	<b>EV3 senzory</b>	<b>28</b>
5.1	Tlačítko . . . . .	28
5.2	Světelný senzor . . . . .	29
5.3	Ultrazvukový dálkoměr . . . . .	30
5.4	Infračervený senzor . . . . .	31
5.5	Gyroskop . . . . .	32

<b>6</b>	<b>Open-Cube senzory</b>	<b>33</b>
6.1	Světelný senzor . . . . .	33
6.2	Laserový dálkoměr . . . . .	35
<b>7</b>	<b>Servomotor</b>	<b>36</b>
<b>8</b>	<b>Parametry a konstanty</b>	<b>38</b>
8.1	Sensor . . . . .	38
8.2	Port . . . . .	39
8.3	Button . . . . .	40
8.4	Light . . . . .	40
8.5	GyroAcc . . . . .	41
8.6	Color . . . . .	41
<b>9</b>	<b>Užitečné funkce MicroPythonu</b>	<b>42</b>
9.1	Výpis informací . . . . .	42
9.2	Generování náhodných čísel . . . . .	42
9.3	Čas . . . . .	43
9.4	Manipulace s binárními daty . . . . .	44
9.5	Vícejádrové programování . . . . .	44
<b>10</b>	<b>Použité pojmy Pythonu</b>	<b>45</b>

## Seznam kódů

1	Použití globální třídy robot. . . . .	6
2	Zjištění stavu tlačítek. . . . .	7
3	Použití LED. . . . .	8
4	Použití bzučáku. . . . .	9
5	Zjištění napětí baterií. . . . .	9
6	Použití displeje. . . . .	13
7	Čtení dat z gyroskopu s využitím irq. . . . .	14
8	Komunikace mezi mikroprocesorem a ESP32. . . . .	19
9	Komunikace s kostkou z Matlabu. . . . .	20
10	Použití NXT tlačítka. . . . .	25
11	Použití NXT světelného senzoru. . . . .	27
12	Použití NXT ultrazvukového senzoru. . . . .	27
13	Použití NXT zvukového senzoru. . . . .	28
14	Použití EV3 tlačítka. . . . .	29
15	Použití EV3 světelného senzoru. . . . .	30
16	Použití EV3 ultrazvukového dálkoměru. . . . .	31
17	Použití EV3 infračerveného senzoru. . . . .	31
18	Použití EV3 gyroskopu. . . . .	33
19	Použití Open-Cube světelného senzoru. . . . .	34
20	Použití Open-Cube laserového dálkoměru. . . . .	35
21	Použití motorů. . . . .	38
22	Výpis informací. . . . .	42
23	Použití knihovny random. . . . .	42
24	Použití knihovny time. . . . .	43
25	Použití třídy Timer. . . . .	43
26	Použití knihovny struct. . . . .	44
27	Použití knihovny _thread. . . . .	44

# 1 Instalace firmwaru

MicroPython firmware spolu se všemi Open-Cube knihovnami popsány v tomto dokumentu je ve výchozím nastavení nahráný v kostce. Aktuální firmware lze také stáhnout z [Open-Cube repozitáře](https://gitlab.fel.cvut.cz/open-cube/firmware/) [ <https://gitlab.fel.cvut.cz/open-cube/firmware/> ] ze složky `micropython`. Pro nahrání firmwaru při jeho aktualizaci, poškození či přepsání postupujte následujícími kroky:

1. Stáhněte si `firmware` [ <https://gitlab.fel.cvut.cz/open-cube/firmware/-/tree/main/micropython/> ] (uf2 binární soubor) pro kostku.
2. Připojte kostku k počítači pomocí USB kabelu.
3. Držte tlačítko `boot select` a stiskněte zapínací tlačítko.
4. V počítači otevřete adresář `RPI-RP2` zkopírujte do něj firmware.
5. Po nahrání firmwaru se kostka restartuje a na displeji se zobrazí menu.

## 2 Editor kódu

Pro úpravu, debugování a nahrávání kódu do kostky doporučujeme použít editor [Thonny](https://thonny.org/) [ <https://thonny.org/> ], který je dostupný pro Windows, Mac i Linux.

### 2.1 Nastavení editoru

Po prvním spuštění editoru klikněte na tlačítko v pravém dolním rohu a v záložce `Interpreter` nastavte interpret na `MicroPython (Raspberry Pi Pico)`. Dále doporučujeme nastavit v záložce `General` možnost `UI mode` na `regular` nebo `expert`. V hlavním okně editoru v záložce `View` zvolte zobrazení lišt `Files` a `Shell`. Lišta `Files` slouží k procházení adresářů v počítači a `Shell` ke komunikaci s kostkou, vypisování informací a chybových hlášek.

### 2.2 Nahrání kódu do kostky

Po propojení zapnuté kostky USB kabelem k počítači se stisknutím tlačítka `Stop/Restart backend` v editoru připojíte ke kostce. V liště `Shell` se zobrazí informace o připojení a v liště `Files` se zobrazí soubory nahrané v kostce. Soubory i celé adresáře můžete kopírovat mezi kostkou a počítačem pravým kliknutím na daný soubor či adresář. Obdobně můžete mazat či vytvářet nové soubory a adresáře.

Upravovat soubory lze jak v adresáři počítače, tak v adresáři kostky. Pokud si otevřete soubor v kostce a po úpravách ho uložíte, automaticky se nahraje do kostky.

Uživatelské programy nahrávejte do adresáře `programs` v kostce. Programem může být jeden soubor s koncovkou `.py` nebo adresář, ve kterém je uživatelský soubor `main.py`. Takto nahrané programy se zobrazí v menu na kostce. Zobrazené jméno programu je určeno názvem souboru bez koncovky `.py` nebo názvem adresáře.

Po úpravě kódu v editoru můžete stisknutím tlačítka `Run current script` spustit na kostce kód aktuálně zobrazený v editoru. Doporučujeme takto spouštět pouze hlavní program `main.py`, který obsahuje framework kostky s inicializací všech potřebných funkcí

pro ovládání periferií. Po spuštění hlavního programu se na kostce zobrazí menu ovládané tlačítky na kostce s možností spuštění nahraného uživatelského programu. Soubor `main.py` se spustí automaticky při zapnutí kostky.

### 3 Robot

Po zapnutí kostky je hlavním programem `main.py` inicializovaný globální objekt `robot`, který obsahuje všechny funkce pro inicializaci, deinicializaci a přístup k objektům periferií kostky. Funkce jsou popsány v následujících kapitolách. Struktura proměnných objektu `robot`, kterými lze přistupovat k objektům periferií je následující:

```
robot
├── battery
├── buttons
├── buzzer
├── display
├── esp
├── led
├── motors[4]
├── sensors
│   ├── ultra_nxt
│   ├── gyro_acc
│   ├── touch[4]
│   ├── light[4]
│   ├── sound[4]
│   ├── ultrasonic[4]
│   ├── gyro[4]
│   ├── infrared[4]
│   └── laser[4]
```

#### `class Robot()`

Inicializace, deinicializace a správa objektů motorů, senzorů a periferií kostky. Automaticky inicializuje tlačítka, LED, bzučák a měření napětí baterií s ochranou podvybití. Uživatelem lze inicializovat senzory, motory a ESP32 Bluetooth komunikaci.

**Inicializace:** Zapnutí kostky.

**Deinicializace:** Vypnutí kostky.

**Přístup:** `robot`

```
init_sensor(sensor_type=None, port=None)
```

Inicializace senzorů.

**Parametry:** `sensor_type` (`Sensor`) – Typ senzoru.

`port` (`Port`) – Port senzoru. Pro typ `Sensor.GYRO` a `Sensor.ULTRA` není potřeba specifikovat.

```
deinit_sensor(sensor_type=None, port=None)
```

Deinicializace senzorů.



**Parametry:** `sensor_type` (`Sensor`) – Typ senzoru. Pokud není specifikován, deinitializuje senzor na daném portu.  
`port` (`Port`) – Port senzoru. Pokud není specifikován, deinitializuje všechny senzory daného typu.

```
init_motor(port=None)
```

Inicializace motorů.

**Parametry:** `port` (`Port`) – Port motoru.

```
deinit_motor(port=None)
```

Deinitializace motorů. Vypne regulátor, snímání enkodérů a zastaví motor.

**Parametry:** `port` (`Port`) – Port motoru.

Následující kód ukazuje použití třídy `robot` na jednoduchém programu pro blikání LED na kostce. Po spuštění tohoto programu z menu se periodicky každou sekundu mění stav LED na kostce. Po stisknutí levého tlačítka se program ukončí, na displeji se opět zobrazí menu a lze spustit další program.

```
1 # Importování funkce pro uspání programu
2 from time import sleep
3 # Importování konstant tlačítek na kostce
4 from lib.robot_consts import Button
5
6 # Definice programu
7 def main:
8     # Přístup ke globalní proměnné robot
9     global robot
10
11     # Smyčka čekající na ukončení programu
12     while True:
13         # Změna stavu LED
14         robot.led.toggle()
15
16         # Získání stavu tlačítek
17         buttons = robot.buttons.pressed()
18         # Ukončení programu, pokud je levé tlačítko stisknuté
19         if buttons[Button.LEFT]:
20             break
21
22         # Uspání programu na 1 sekundu
23         sleep(1)
24
25 # Spuštění programu
26 main()
```

Kód 1: Použití globální třídy `robot`.

## 3.1 Tlačítka

```
class Buttons()
```

Informace o stavu stisknutí tlačítek na čelním panelu kostky. Pokud se po krátkém stisknutí tlačítka POWER kostka sama nevypne, lze ji vypnout dlouhým stisknutím tohoto tlačítka, čímž dojde k odpojení kostky od napájení.

**Inicializace:** Zapnutí kostky.

**Deinicializace:** Vypnutí kostky.

**Přístup:** robot.buttons

```
pressed()
```

Navrátí stav tlačítek. Pro zjištění stavu konkrétního tlačítka lze použít [konstanty tlačítek](#).

**Vrací:** Tuple stavu tlačítek zapnout, doleva, doprava, ok, nahoru, dolů. True pokud je tlačítko stisknuté, False pokud není.

**Typ:** (bool, bool, bool, bool, bool, bool)

```
1 # Importování funkce pro uspání programu
2 from time import sleep
3 # Importování konstant tlačítek na kostce
4 from lib.robot_consts import Button
5
6 def main:
7     global robot
8     while True:
9         # Získání stavu tlačítek
10        buttons = robot.buttons.pressed()
11
12        # Zobrazení stavu tlačítek v terminálu
13        print("Tlačítko stisknuté:",
14              "zapnout:", buttons[Button.POWER],
15              "doleva:", buttons[Button.LEFT],
16              "doprava:", buttons[Button.RIGHT],
17              "OK:", buttons[Button.OK],
18              "nahoru:", buttons[Button.UP],
19              "dolů:", buttons[Button.DOWN])
20        # Ukončení programu, pokud je levé tlačítko stisknuté
21        if buttons[Button.LEFT]:
22            break
23        # Uspání programu na 1 sekundu
24        sleep(1)
25
26 # Spuštění programu
27 main()
```

Kód 2: Zjištění stavu tlačítek.

## 3.2 LED

```
class Led()
```

Zapínání a vypínání červené LED na čelním panelu kostky.

**Inicializace:** Zapnutí kostky.

**Deinicializace:** Vypnutí kostky.

**Přístup:** `robot.led`

```
on()
```

Zapne LED.

```
off()
```

Vypne LED.

```
toggle()
```

Změní stav LED.

```
1 # Zapnutí LED
2 robot.led.on()
3 # Vypnutí LED
4 robot.led.off()
```

Kód 3: Použití LED.

## 3.3 Bzučák

```
class Buzzer()
```

Ovládání bzučáku v kostce. Bzučák má nejvyšší hlasitost při frekvenci přibližně 4500 Hz z důvodu nerovnoměrné frekvenční charakteristiky piezo měniče. Hlasitost lze dále nastavit otočením odporového trimru na čelním panelu kostky.

**Inicializace:** Zapnutí kostky.

**Deinicializace:** Vypnutí kostky.

**Přístup:** `robot.buzzer`

```
set_freq_duty(freq, duty)
```

Nastaví PWM ovládající bzučák na požadovanou frekvenci a střidu.

- Parametry:**
- **freq** (frekvence: Hz) – Frekvence PWM.
  - **duty** (střída: %) – Střída PWM.

```
off()
```

Vypne bzučák.

```

1 # Zapnutí bzučáku na frekvenci 4000 Hz se střídou 50 %
2 robot.buzzer.set_freq_duty(4000, 50)
3 # Změna frekvence bzučáku na 1000 Hz
4 robot.buzzer.set_freq_duty(1000, 50)
5 # Vypnutí bzučáku
6 robot.buzzer.off()

```

Kód 4: Použití bzučáku.

### 3.4 Baterie

#### `class Battery()`

Měření napětí na napájecích bateriích. Při inicializaci se nastaví časovač s periodou 200 ms, který spouští měření napětí.

Open-Cube je napájena dvěma Li-ion akumulátory (nominální napětí jednoho článku je 3,7 V, maximální nabíjecí 4,2 V, vybití pod 3 V již zkracuje životnost článku). V případě plného nabití lze tedy naměřit 8,2–8,4 V. Akumulátory by neměly být vybíjeny pod 6 V. Kostka se softwarově vypne při poklesu napětí pod 6,5 V, hardwarová podpěťová ochrana by měla články odpojit při poklesu pod 5,8 V.

**Inicializace:** Zapnutí kostky.

**Deinicializace:** Vypnutí kostky.

**Přístup:** `robot.battery`

#### `voltage()`

Navrátí poslední změřené napětí na bateriích.

**Vrací:** Poslední změřené napětí na bateriích.

**Typ:** float: V

#### `read_voltage()`

Změří napětí na bateriích.

**Vrací:** Poslední změřené napětí na bateriích.

**Typ:** float: V

#### `deinit()`

Vypne periodické měření napětí.

```

1 # Zjistí napětí na bateriích ve voltech
2 voltage = robot.battery.voltage()

```

Kód 5: Zjištění napětí baterií.

## 3.5 Displej

**class** SH1106\_I2C()

Zobrazování textu, geometrických tvarů a grafů na displeji. Aktivování jednotlivých pixelů se zaznamenává do frame bufferu, který je příkazem `show` celý přenesen do displeje. Displej má rozlišení 128x64 pixelů.

**Inicializace:** Zapnutí kostky.

**Deinicializace:** Vypnutí kostky.

**Přístup:** `robot.display`.

**show()**

Zobrazí aktuální frame buffer na displeji.

**fill(*color*)**

Vyplní celý frame buffer určenou barvou.

**Parametry:** `color` (0/1) – Barva – 0 černá, 1 bílá.

**pixel(*x*, *y* [, *color*])**

Nastaví pixel ve frame bufferu na určenou barvu. Pokud barva není určena, navrátí nastavenou barvu pixelu.

**Parametry:** • `x`, `y` – Souřadnice pixelu.

• `color` (0/1) – Barva – 0 černá, 1 bílá.

**Vrací:** Barva pixelu.

**Typ:** `int` (0/1)

**hline(*x*, *y*, *w*, *color*)**

Nakreslí horizontální linii do frame bufferu.

**Parametry:** • `x`, `y` – Souřadnice levého začátku linie.

• `w` – Šířka linie.

• `color` (0/1) – Barva – 0 černá, 1 bílá.

**vline(*x*, *y*, *h*, *color*)**

Nakreslí vertikální linii do frame bufferu.

**Parametry:** • `x`, `y` – Souřadnice horního začátku linie.

• `h` – Výška linie.

• `color` (0/1) – Barva – 0 černá, 1 bílá.

**line(*x1*, *y1*, *x2*, *y2*, *color*)**

Nakreslí linii do frame bufferu.

- Parametry:**
- **x1, y1** – Souřadnice začátku linie.
  - **x2, y2** – Souřadnice konce linie.
  - **color** (0/1) – Barva – 0 černá, 1 bílá.

**rect**(*x, y, w, h, color*)

Nakreslí obdélník do frame bufferu.

- Parametry:**
- **x, y** – Souřadnice levého horního vrcholu obdélníku.
  - **w, h** – Výška a šířka obdélníku.
  - **color** (0/1) – Barva – 0 černá, 1 bílá.

**fill\_rect**(*x, y, w, h, color*)

Nakreslí vyplněný obdélník do frame bufferu.

- Parametry:**
- **x, y** – Souřadnice levého horního vrcholu obdélníku.
  - **w, h** – Výška a šířka obdélníku.
  - **color** (0/1) – Barva – 0 černá, 1 bílá.

**ellipse**(*x, y, xr, yr, color, f=False, m=1111b*)

Nakreslí elipsu do frame bufferu.

- Parametry:**
- **x, y** – Souřadnice středu elipsy.
  - **xr, yr** – Délka poloos elipsy.
  - **color** (0/1) – Barva – 0 černá, 1 bílá.
  - **f** (bool) – Vyplnění elipsy pokud je parametr specifikován a má hodnotu True.
  - **m** (4 bity) – Omezení vykreslení elipsy do určených kvadrantů. Bit 0 určuje Q1, b1 Q2, b2 Q3 a b3 Q4. Kvadranty jsou číslovány proti směru hodinových ručiček a Q1 je pravý horní kvadrant.

**fill\_rect**(*x1, y1, x2, y2, x3, y3, color*)

Nakreslí vyplněný trojúhelník do frame bufferu.

- Parametry:**
- **x1, y1, x2, y2, x3, y3** – Souřadnice vrcholů trojúhelníku.
  - **color** (0/1) – Barva – 0 černá, 1 bílá.

**text**(*s, x, y, color=1*)

Nakreslí text do frame bufferu. Znaky mají rozměr 8x8 pixelů.

- Parametry:**
- **s** (str) – Text.
  - **x, y** – Souřadnice levého horního rohu začátku textu.
  - **color** (0/1) – Barva – 0 černá, 1 bílá.

**centered\_text**(*s, y, color=1*)

Nakreslí vycentrovaný text do frame bufferu. Znaky mají rozměr 8x8 pixelů.

- Parametry:**
- **s** (str) – Text.
  - **x** – Vertikální souřadnice levého horního rohu začátku textu.
  - **color** (0/1) – Barva – 0 černá, 1 bílá.

```
draw_bar_chart_v(value, x, y, w, h, low_lim=0, high_lim=100, no_of_tics=5, label=None, redraw=False)
```

Nakreslí vertikální sloupcový graf do frame bufferu.

**Vrací:** Překreslení grafu pro snížení problikávání grafu.

**Typ:** bool

- Parametry:**
- **value** (float) – Zobrazovaná hodnota.
  - **x, y** (int) – Souřadnice levého dolního rohu začátku grafu.
  - **w, h** (int) – Šířka a výška grafu.
  - **low\_lim, high\_lim** (int) – Dolní a horní omezení grafu.
  - **no\_of\_tics** (int) – Počet rozdělení grafu pro lepší čitelnost. Doporučená maximální hodnota je 5.
  - **label** (str) – Popisek grafu.
  - **redraw** (bool) – Překreslení grafu.

```
draw_bar_chart_h(value, x, y, w, h, low_lim=0, high_lim=100, no_of_tics=5, label=None, redraw=False)
```

Nakreslí horizontální sloupcový graf do frame bufferu.

**Vrací:** Překreslení grafu pro snížení problikávání grafu.

**Typ:** bool

- Parametry:**
- **value** (float) – Zobrazovaná hodnota.
  - **x, y** (int) – Souřadnice levého dolního rohu začátku grafu.
  - **w, h** (int) – Šířka a výška grafu.
  - **low\_lim, high\_lim** (int) – Dolní a horní omezení grafu.
  - **no\_of\_tics** (int) – Počet rozdělení grafu pro lepší čitelnost. Doporučená maximální hodnota je 5.
  - **label** (str) – Popisek grafu.
  - **redraw** (bool) – Překreslení grafu.

```
draw_dial(value, x, y, r, loval, hival, no_of_steps, label, redraw)
```

Nakreslí kruhový číselník do frame bufferu.

**Vrací:** Překreslení grafu pro snížení problikávání číselníku.

**Typ:** bool

- Parametry:**
- **value** (float) – Zobrazovaná hodnota.
  - **x, y** (int) – Souřadnice středu číselníku.
  - **r** (int) – Poloměr číselníku.
  - **loval, hival** (int) – Dolní a horní omezení číselníku.
  - **no\_of\_steps** (int) – Počet rozdělení číselníku. Doporučená maximální hodnota je 10.
  - **label** (str) – Popisek číselníku.
  - **redraw** (bool) – Překreslení číselníku.

**continuous\_graph**(*x, y, gx, gy, w, h, xlo, xhi, ylo, yhi, label, redraw*)

Nakreslí průběžný graf do frame bufferu. Funkce si vnitřně udržuje hodnoty grafu. Při vyplnění šířky grafu se předchozí hodnoty odstraní a graf se začne vykreslovat od počátku.

**Vrací:** Překreslení grafu pro snížení problikávání grafu.

**Typ:** bool

**Parametry:**

- **x, y** (float) – Zobrazovaná souřadnice v grafu.
- **gx, gy** (int) – Souřadnice levého dolního rohu grafu.
- **w, h** (int) – Šířka a výška grafu.
- **xlo, xhi** (int) – Dolní a horní omezení osy x.
- **ylo, yhi** (int) – Dolní a horní omezení osy y.
- **label** (str) – Popisek grafu.
- **redraw** (bool) – Překreslení grafu.

```
1 # Vymazání displeje
2 robot.display.fill(0)
3 # Zapsání textu do frame bufferu do levého horního rohu
4 robot.display.text("test message", 0, 0, 1)
5
6 # Zapsání čísel s přesností na 2 desetinná místa do frame bufferu na
7   další řádek
8 x, y = 1.23, 56.789
9 robot.display.text(f"x={x:.2f}, y: {y:.2f}", 0, 10, 1)
10 # Zobrazení frame bufferu na displeji
11 robot.display.show()
12
13 # Funkce, které vrací hodnotu pro snížení problikávání grafu by měly
14   být ve smyčce volány následovně
15 redraw = True
16 while(True):
17     redraw = robot.display.draw_dial(50, 128/2, 40, 20, 0, 100, 10, 270,
18     "Speedometer", redraw)
```

Kód 6: Použití displeje.

## 3.6 Gyroskop a akcelerometr

**class** ICM42688()

Integrovaný gyroskop a akcelerometr.

Měření zrychlení a úhlové rychlosti ve třech osách. Interrupt pin senzoru je připojen na [GyroAcc.IRQ\\_PIN](#) (GPIO 28) mikrokontroléru.

**Inicializace:** robot.init\_sensor(sensor\_type=[Sensor.GYRO\\_ACC](#))

**Deinicializace:** robot.deinit\_sensor(sensor\_type=[Sensor.GYRO\\_ACC](#))

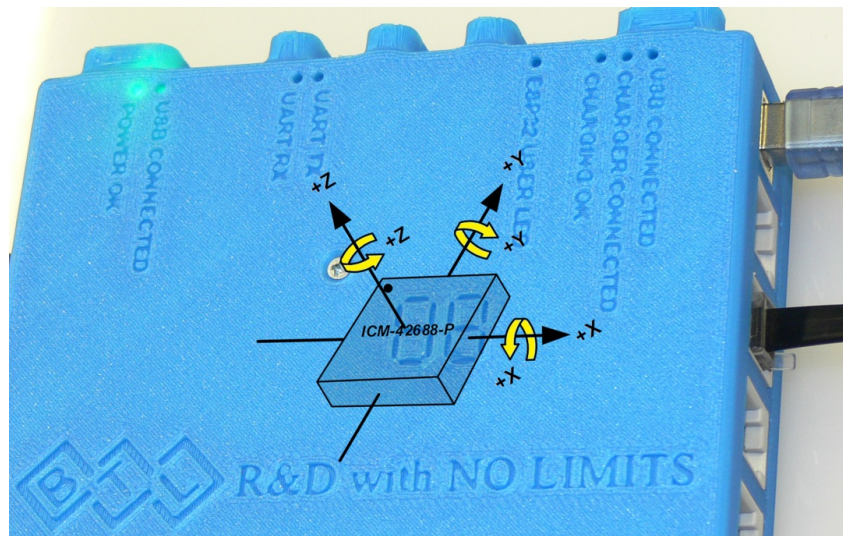
**Přístup:** robot.sensors.gyro\_acc

**read\_value()**

Přečte ze senzoru a navrátí změřené hodnoty úhlových rychlostí a zrychlení. Pro zjištění konkrétní hodnoty lze použít [GyroAcc](#) konstanty gyroskopu a akcelerometru.



**Vrací:** Tuple zrychlení a úhlových rychlostí v osách x, y, z.  
**Typ:** (a\_x: -, a\_y: -, a\_z: -, g\_x: °/s, g\_y: °/s, g\_z: °/s)



Obrázek 1: Orientace os integrovaného gyroskopu a akcelerometru.

Příklad čtení dat z gyroskopu s využitím irq (pinu přerušení):

```
1 # Importování knihoven
2 from time import sleep
3 from machine import Pin
4 # Importování konstant
5 from lib.robot_consts import Button, Sensor, GyroAcc
6
7 a_g_data = (0,0,0,0,0,0)
8
9 def main:
10     global robot, a_g_data
11     # Inicializace senzoru
12     robot.init_sensor(sensor_type=Sensor.GYRO_ACC)
13     # Nastavení GPIO 28
14     icm_irq_pin = Pin(GyroAcc.IRQ_PIN, Pin.IN)
15     # Nastavení callback funkce volané na náběžnou hranu irq, pokud jsou
16     # nová data k dispozici
17     icm_irq_pin.irq(trigger=Pin.IRQ_RISING, handler=callback)
18
19 # Smyčka čekající na ukončení programu
20 while True:
21     # Zobrazení posledních hodnot zrychlení a úhlové rychlosti
22     print("Acc:", a_g_data[GyroAcc.AX],
23           a_g_data[GyroAcc.AY],
24           a_g_data[GyroAcc.AZ],
25           "Gyro:", a_g_data[GyroAcc.GX],
26           a_g_data[GyroAcc.GY],
27           a_g_data[GyroAcc.GZ],)
28     # Získání stavu tlačítek
29     buttons = robot.buttons.pressed()
30     # Ukončení programu, pokud je levé tlačítko stisknuté
31     if buttons[Button.LEFT]:
32         break
```

```

32     # Uspání programu na 1 sekundu
33     sleep(1)
34
35     # Zrušení callbacku před deinitializací senzoru
36     icm_irq_pin.irq(handler=None)
37     # Deinitializace senzoru
38     robot.deinit_sensor(sensor_type=Sensor.GYRO_ACC)
39
40 def callback(p):
41     global robot, a_g_data
42     # Přečtení nových hodnot ze senzoru
43     a_g_data = robot.sensors.gyro_acc.read_value()
44
45 # Spuštění programu
46 main()

```

Kód 7: Čtení dat z gyroskopu s využitím irq.

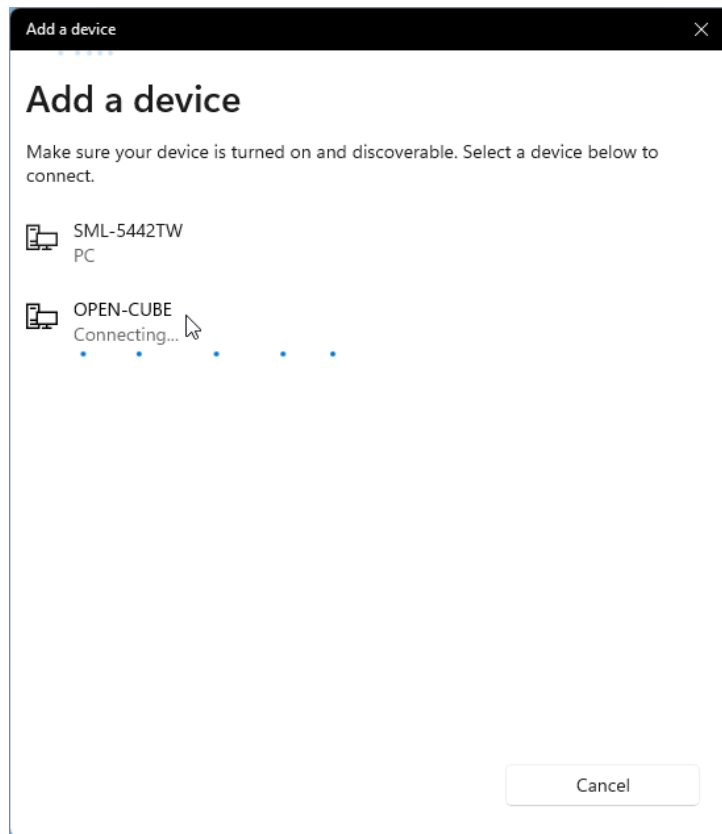
## 3.7 ESP32 komunikace

ESP32 komunikuje s kostkou (RP2040) pomocí sériové linky. S dalším zařízením může komunikovat pomocí Wi-Fi nebo Bluetooth. ESP je programovatelné po připojení USB kabelem na vrchní straně kostky. Výchozí firmware ESP (dostupný z [Open-Cube repozitáře](https://gitlab.fel.cvut.cz/open-cube/firmware/-/tree/main/ESP) [ <https://gitlab.fel.cvut.cz/open-cube/firmware/-/tree/main/ESP> ]) umožňuje v Bluetooth režimu přeposílání dat z kostky přes Bluetooth virtuální COM port počítači nebo mobilu a ve Wi-Fi režimu funguje jako access point s web serverem s univerzálními ovládacími prvky (tlačítka, přepínače, indikátory).

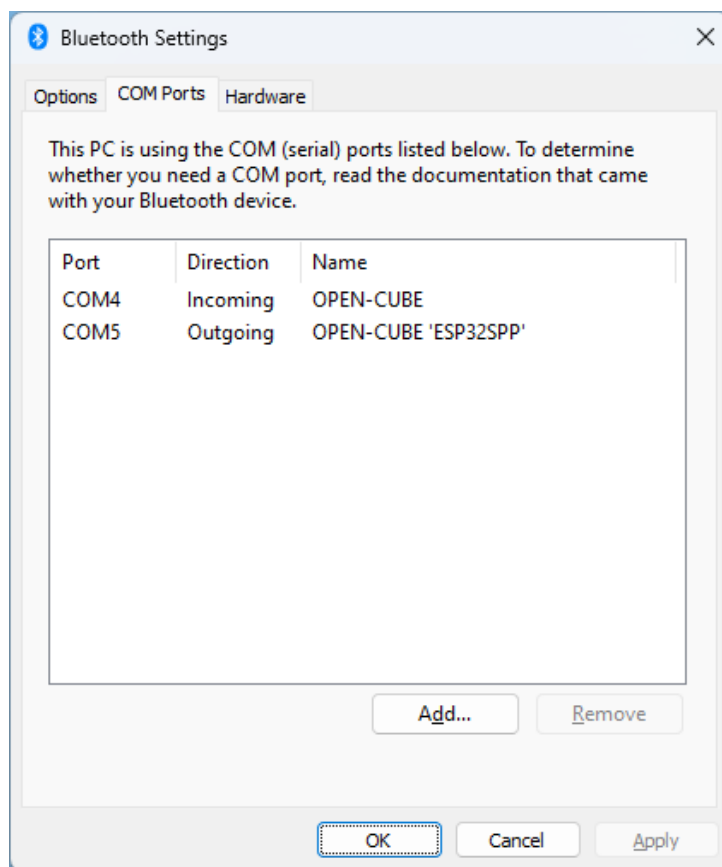
### 3.7.1 Bluetooth propojení ESP32 s jiným zařízením

Komunikace probíhá sériově přes Bluetooth virtuální COM port a je otestována pouze pro Windows a Android.

1. Pro spárování musí být kostka zapnutá v menu.
2. Ve Windows Bluetooth nastavení zvolte Přidat nové zařízení, dále Bluetooth a spárujte počítač s kostkou. Viz obrázek 2.
3. Potvrďte pin na kostce a na počítači.
4. Pro zjištění čísla COM portu pokračujte do pokročilých nastavení Bluetooth. Požadovaný COM port má název obsahující ESP32SPP a směr Outgoing. Viz obrázek 3. Tento COM port lze použít pro přijímání i odesílání dat.



Obrázek 2: Spárování počítače s kostkou.



Obrázek 3: Zjištění čísla COM portu.

### 3.7.2 Kostka

```
class ESP()
```

Oboustranné posílání dat mezi mikrokontrolérem a ESP32 pomocí rozhraní UART. Při inicializaci se spustí časovač s periodou 10 ms, při kterém se přijímají a ukládají data posílaná ESP.

ESP může pracovat ve dvou režimech – Bluetooth a Wi-Fi access point.

1. V případě Bluetooth režimu se data posílají po zprávách, což jsou celky dat patřící k sobě. Odesílání a přijímání dat může probíhat v ASCII nebo v binárním režimu.

ASCII režim: Odesílaná zpráva je zakončena terminátorem LF (new line). Při přijímání dat je zpráva k dispozici po přijetí terminátoru LF nebo CR. Data jsou typu string.

Binární režim: Pro komunikaci je nutné specifikovat hlavičku odesílaných a přijímaných dat, aby bylo možné rozeznat začátek zprávy. Aby bylo možné zprávu správně přijmout a určit její konec, musí být předem specifikovaná velikost zprávy v bytech. Data jsou typu bytes a jejich interpretace je ponechána na uživateli.

2. Wi-Fi access point režim s web serverem poskytuje univerzální ovládací prvky – tlačítka, přepínače a indikátory.

**Inicializace:** Zapnutí kostky.

**Deinicializace:** Vypnutí kostky.

**Přístup:** `robot.esp`

```
reset()
```

Resetuje ESP a nastaví Bluetooth režim.

```
bt()
```

Nastaví Bluetooth režim.

```
wifi()
```

Nastaví Wi-Fi access point režim. Výchozí adresa web serveru je 192.168.4.1. Indexace prvků web serveru je zobrazena na obrázku 4.

```
set_name(name)
```

Nastaví jméno ESP pro Bluetooth a Wi-Fi. Výchozí jméno je "OPEN-CUBE".

**Parametry:** **name** (str) – Jméno ESP.

```
set_password(password)
```

Nastaví heslo ESP pro Wi-Fi. Heslo musí mít alespoň 8 znaků. Výchozí heslo je "12345678".

**Parametry:** **password** (str) – Heslo ESP.

### **bt\_read()**

Navrátí poslední přijatou zprávu v Bluetooth režimu.

**Vrací:** Přijatá zpráva. Pokud není nová zpráva k dispozici: None.

**Typ:**

- ASCII režim: str
- Binární režim: bytes

### **bt\_write(*buff*)**

Odešle zprávu v Bluetooth režimu.

**Parametry:** **buff** (buffer: str(ASCII)/bytes(binární)) – Buffer odesílaných dat.

### **bt\_set\_binary(*header, data\_size*)**

Bluetooth režimu nastaví komunikaci binárního režimu. Pokud je header None, komunikace se nastaví do ASCII režimu.

**Parametry:** **header** (tuple) – Hlavička libovolné délky obsahující čísla 0–255, nebo prvek None.

**data\_size** (int) – Velikost přijímané zprávy v bytech.

### **wifi\_get\_buttons()**

Navrátí stav stisknutí 9 tlačítek web serveru.

**Vrací:** Tuple stavu tlačítek.

**Typ:** 9x bool

### **wifi\_get\_switches()**

Navrátí stav 5 přepínačů web serveru.

**Vrací:** Tuple stavu přepínačů.

**Typ:** 5x bool

### **wifi\_set\_indicators(*indicators*)**

Nastaví stav 5 indikátorů web serveru.

**Parametry:** **indicators** (tuple) – 5x bool stavu indikátorů.

### **wifi\_set\_numbers(*numbers*)**

Nastaví 6 čísel web serveru.

**Parametry:** **indicators** (tuple) – 6x float.

### **wifi\_set\_buttons\_labels(*labels*)**

Nastaví popisek 9 tlačítek web serveru. String jednoho tlačítka může mít délku maximálně 9 znaků a prázdný string skryje dané tlačítko.

**Parametry:** **labels** (tuple) – 9x str popisků tlačítek.

### **wifi\_set\_switches\_labels(*labels*)**

Nastaví popisek 5 přepínačů web serveru. String jednoho přepínače může mít délku maximálně 9 znaků a prázdný string skryje daný přepínač.

**Parametry:** `labels` (tuple) – 5x str popisků přepínačů.

#### `wifi_set_indicators_labels(labels)`

Nastaví popisek 5 indikátorů web serveru. String jednoho indikátoru může mít délku maximálně 9 znaků a prázdný string skryje daný indikátor.

**Parametry:** `labels` (tuple) – 5x str popisků indikátorů.

#### `wifi_set_numbers_labels(labels)`

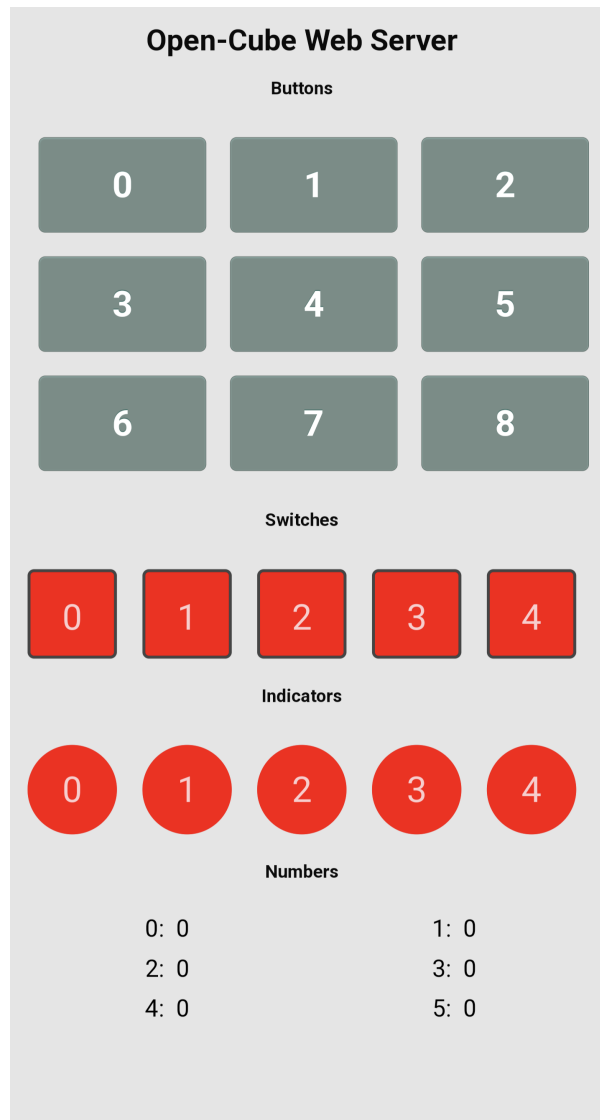
Nastaví popisek 6 čísel web serveru. String jednoho čísla může mít délku maximálně 9 znaků a prázdný string skryje dané číslo.

**Parametry:** `labels` (tuple) – 6x str popisků čísel.

Příklad použití komunikace mezi mikroprocesorem a ESP32:

```
1 # Importování knihovny pro práci s binárními daty
2 import struct
3
4 robot.esp_uart_init()
5 # Odeslání a přijetí zprávy typu string v režimu ASCII
6 robot.esp.bt_write("test message")
7 received_message = robot.esp.bt_read()
8
9 # Odeslání a přijetí zprávy o velikosti 8 bytů v binárním režimu
10 robot.esp.bt_set_binary((112, 241, 3, 62), 8)
11 # Vytvoření a odeslání zprávy typu bytes se dvěma čísly typu single
12 write_message = struct.pack("<ff", 521.9, -11.821)
13 robot.esp.bt_write(write_message)
14 # Příjem a dekodování zprávy obsahující dvě čísla typu single
15 received_message2 = robot.esp.bt_read()
16 (value1, value2) = struct.unpack("<ff", received_message2)
17
18 # Přepne ESP do Wi-Fi access point režimu
19 robot.esp.wifi()
20 # Nastaví popisky indikátorů
21 robot.esp.set_indicators_labels(["1", "2", "3", "4", "5"])
22 # Zjistí stav přepínačů
23 switches = robot.esp.get_switches()
```

Kód 8: Komunikace mezi mikroprocesorem a ESP32.



Obrázek 4: Indexace prvků ESP web serveru.

### 3.7.3 Serial Bluetooth Terminal

Pro jednoduchou komunikaci z mobilu lze použít aplikaci [Serial Bluetooth Terminal](https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal) [ [https://play.google.com/store/apps/details?id=de.kai\\_morich.serial\\_bluetooth\\_terminal](https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal) ].

### 3.7.4 Matlab

Pro komunikaci pomocí Matlabu je zapotřebí mít nainstalovaný Communications Toolbox. Toto rozhraní používá Bluetooth SPP a umožňuje přijímat a odesílat ASCII a binární data. Při posílání ASCII dat je zpráva zakončena terminátorem. V případě binárních dat zpráva není zakončena terminátorem a kostka i program v Matlabu musí předem znát délku zprávy, aby mohla být správně interpretovaná.

Příklad použití rozhraní je uveden v následujícím kódu. Další informace k použití rozhraní naleznete v [oficiální dokumentaci](https://www.mathworks.com/help/matlab/bluetooth-communication.html) [ <https://www.mathworks.com/help/matlab/bluetooth-communication.html> ].

```

1 % Propojení kostky s Matlabem.
2 cube = bluetooth('OPEN-CUBE')
3 % Nastavená terminátoru LF (new line) pro odesílání a příjem
  ASCII dat.
4 configureTerminator(device, 'LF')
5 %%
6 % Poslání testovací zprávy do kostky. Zpráva je zakončena
  nastaveným terminátorem.
7 writeline(cube, 'test message')
8 % Příjmutí zprávy z kostky. Funkce přijímá data a čeká na
  zakončení nastaveným terminátorem.
9 cube_message = readline(cube)
10 %%
11 % Poslání testovací zprávy do kostky. Zpráva je
  interpretovaná jako formát string a není zakončena
  terminátorem.
12 write(cube, 'test message', 'string')
13 % Příjmutí zprávy z kostky. Zpráva má délku 8 bytů a je
  interpretovaná jako formát float.
14 cube_message2 = read(cube, 8, 'float')

```

Kód 9: Komunikace s kostkou z Matlabu.

### 3.7.5 Simulink

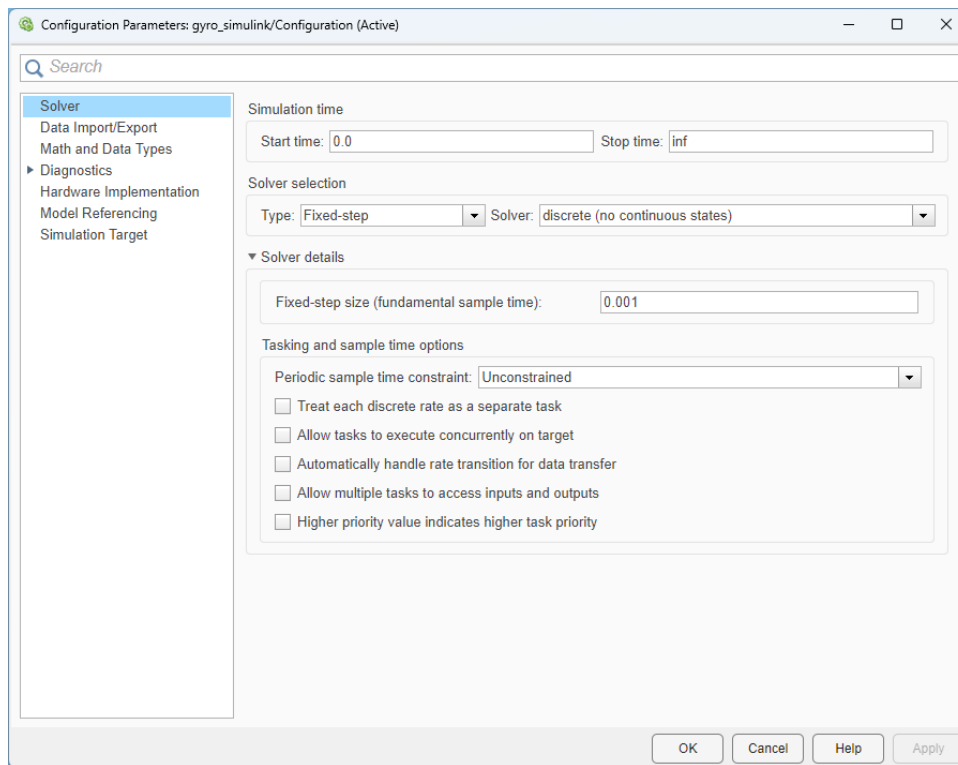
Pro komunikaci pomocí Simulinku je zapotřebí mít nainstalovaný Instrument Control Toolbox, který poskytuje bloky Serial Configuration, Serial Send a Serial Receive pro sériovou komunikaci. Při vytvoření nového modelu je potřeba upravit nastavení řešiče simulace a nastavit parametry sériové komunikace:

1. Kliknutím na tlačítko v pravém dolním rohu nastavit řešič na Fixed-Step, discrete a Fixed-step size na hodnotu podle frekvence odesílání a přijímání dat (např. 0,001). Viz obrázek 5
2. Kliknutím na šipku pod tlačítkem Run zapnout Simulation Pacing, viz obrázky 6 a 7.
3. V modelu vytvořit blok Serial Configuration, ve kterém nastavit COM port, zjištěný v kapitole 3.7.1 a ostatní parametry podle obrázku 8.

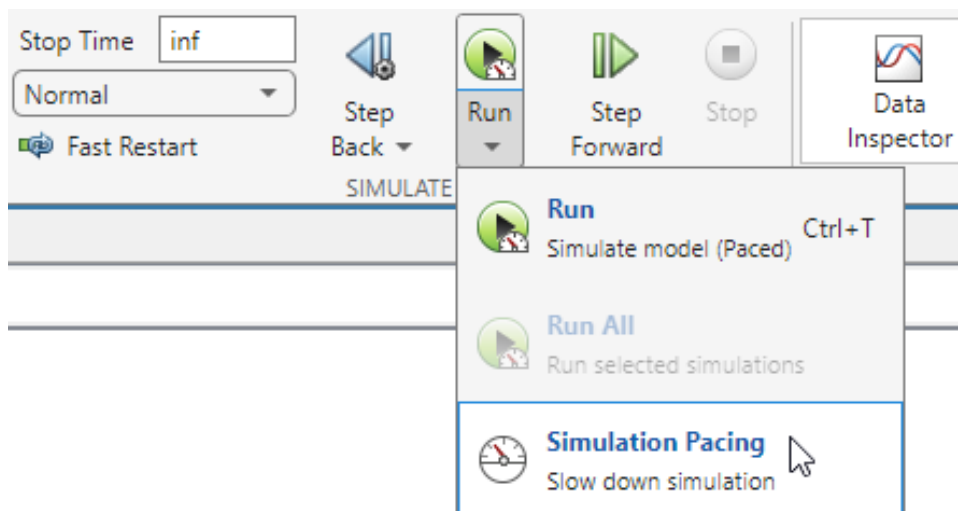
Po tomto nastavení je možné použít bloky Serial Send a Serial Receive pro odesílání a přijímání dat. Informace k použití bloků naleznete v [oficiální dokumentaci](https://www.mathworks.com/help/instrument/direct-interface-communication-in-simulink.html) [ <https://www.mathworks.com/help/instrument/direct-interface-communication-in-simulink.html> ].

Příklad použití sériové komunikace pro nastavení konstant PID regulátoru v kostce a zobrazení aktuálních složek PID je na obrázku 9. V tomto příkladu se hodnoty odesílají a přijímají v binárním formátu, je nastavena hlavička pro odesílání a přijímání a data jsou typu single (4 byty). Data se přijímají každé 0,01 sekundy, což je nastaveno v bloku Serial Send v kolonce Block sample time. Data se odesílají každou sekundu, což je nastaveno v kolonkách Block sample time tří bloků Constant. Nastavený Fixed-step size řešiče by měl být menší než Block sample time.

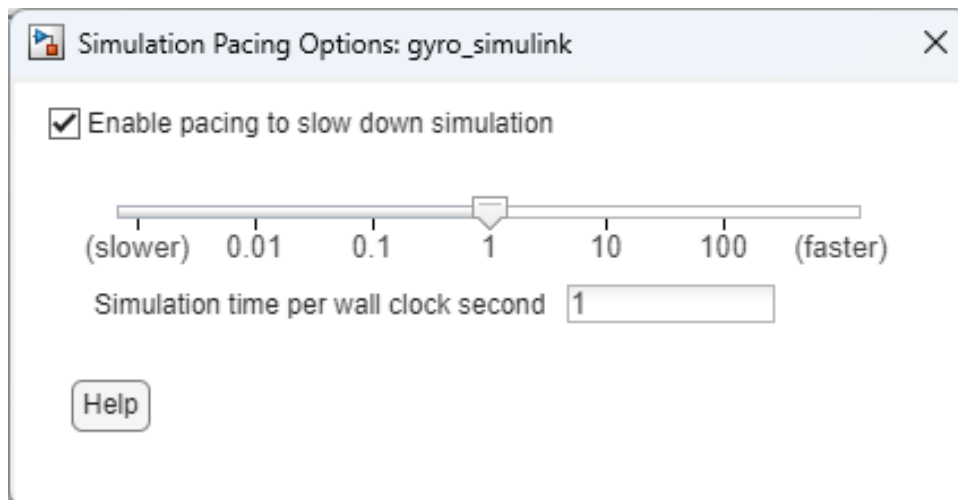




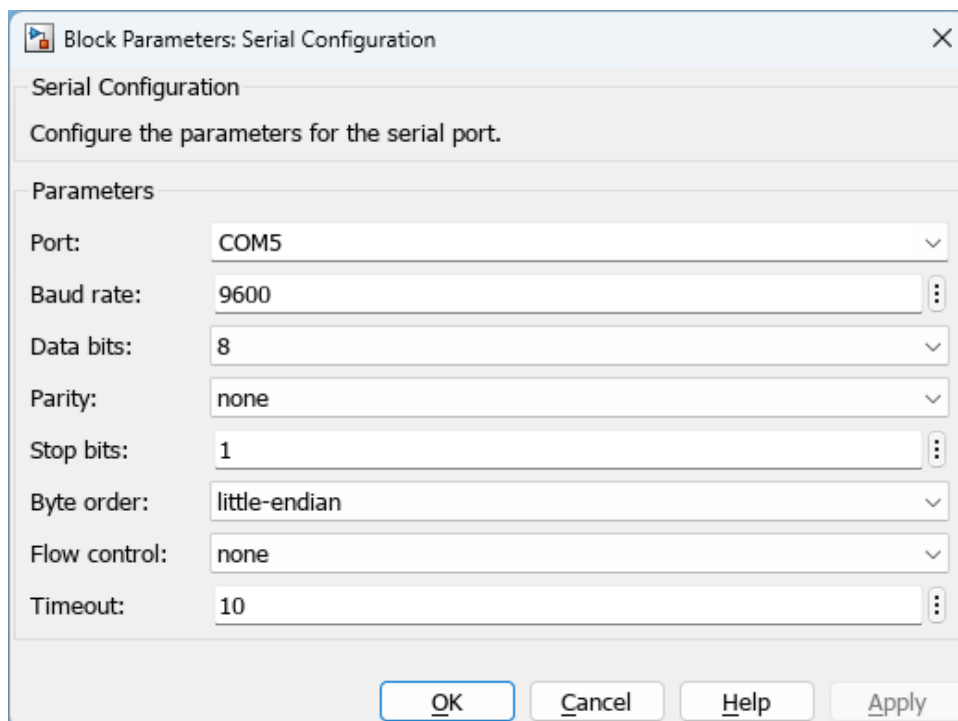
Obrázek 5: Nastavení řešiče Simulinku.



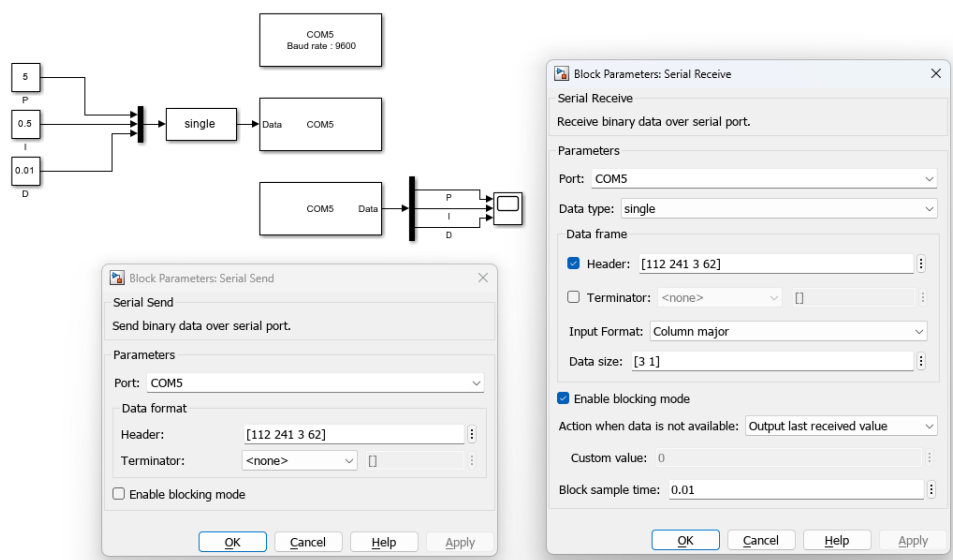
Obrázek 6: Umístění nastavení Simulation pacing v Simulinku.



Obrázek 7: Nastavení Simulation Pacing.



Obrázek 8: Nastavení bloku Serial Configuration v Simulinku.



Obrázek 9: Příklad použití sériové komunikace v Simulinku.

## 4 NXT senzory

### 4.1 Tlačítko

```
class TouchSensor(port)
```

NXT Touch Sensor.

Informace o stavu stisknutí tlačítka.

**Parametry:** **port** (**Port**) – Port, ke kterému je senzor připojen.

**Inicializace:** `robot.init_sensor(sensor_type=Sensor.NXT_TOUCH, port=Port)`.

**Deinicializace:** `robot.deinit_sensor(sensor_type=Sensor.NXT_TOUCH, port=Port)`.

**Přístup:** `robot.sensors.touch[Port]`.

```
pressed()
```

Navrátí stav tlačítka.

**Vrací:** True pokud je tlačítko stisknuté, False pokud není.

**Typ:** bool

```
1 # Importování konstant typů senzorů a portů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace NXT tlačítka na senzorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.NXT_TOUCH, port=Port.S1)
6
7 # Zjištění stavu stisknutí tlačítka
8 touch_pressed = robot.sensors.touch[Port.S1].pressed()
```

Kód 10: Použití NXT tlačítka.

### 4.2 Světelný senzor

```
class LightSensor(port)
```

NXT Ligth Sensor.

Měření intenzity dopadajícího světla.

Měření je umožněno dvěma módy – manuální, při kterém je intenzita změřena na požádání a kontinuální, při kterém je intenzita měřena periodicky a na požádání je vrácena poslední změřená hodnota. Kontinuální mód dále poskytuje režimy neblíkající a blikající. Při nastavení neblíkajícího režimu je intenzita měřena ve stavu LED nastaveném uživatelem. V blikajícím režimu je intenzita měřena při vypnutém i zapnutém stavu LED. Změna stavu LED je prováděna automaticky a uživateli je na požádání k dispozici poslední změřená intenzita pro oba stavy LED.

AD převodník vzorkuje signál s periodou 1 ms. Intenzita dopadajícího světla je získána měřením napětí na senzoru. Při změně stavu LED se toto napětí ustálí za přibližně 10 ms.

**Parametry:** `port` (`Port`) – Port, ke kterému je senzor připojen.  
**Inicializace:** `robot.init_sensor(sensor_type=Sensor.NXT_LIGHT, port=Port)`  
**Deinicializace:** `robot.deinit_sensor(sensor_type=Sensor.NXT_LIGHT, port=Port)`  
**Přístup:** `robot.sensors.light[Port]`

### `on()`

Zapne LED.

### `off()`

Vypne LED.

### `togggle()`

Změní stav LED.

### `intensity(pin_on)`

V manuálním módu změří intenzitu světla a navrátí ji. V kontinuálním módu navrátí poslední změřenou intenzitu.

**Parametry:** `pin_on` (bool) – Volba navrácené změřené intenzity dopadajícího světla. V manuálním módu a v kontinuálním módu s neblíkajícím režimem na hodnotě nezáleží. V kontinuálním módu s blikajícím režimem navrátí pro True intenzitu změřenou při zapnuté LED a pro False při vypnuté LED.

**Vrací:** Intenzita dopadajícího světla na senzor. 0 pro největší intenzitu, 32768 pro nejmenší.

**Typ:** int (0–32768)

### `set_continuous(period_us, wait_us)`

Nastaví měření do kontinuálního módu.

**Parametry:** • `period_us` (int) – Doba mezi začátkem konverze analogové hodnoty a vyčtením konvertované digitální hodnoty z AD převodníku. Doporučená minimální hodnota je 1100.

• `wait_us` (int) – Doba čekání před měřením po změně stavu LED v mikrosekundách. Doporučená minimální hodnota je 10000.

### `stop_continuous()`

Nastaví měření do manuálního módu.

### `set_switching()`

Nastaví měření kontinuálního módu do blikajícího režimu.

```
stop_switching()
```

Nastaví měření kontinuálního módu do neblíkajícího režimu.

```
1 # Importování konstant typů senzorů a portů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace NXT světelného senzoru na senzorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.LIGHT, port=Port.S1)
6
7 # Zapnutí LED senzoru
8 robot.sensors.light[Port.S1].on()
9 # Změření intenzity světla
10 light_intensity = robot.sensors.light[Port.S1].intensity()
```

Kód 11: Použití NXT světelného senzoru.

### 4.3 Ultrazvukový dálkoměr

```
class UltrasonicSensor()
```

NXT Ultrasonic Sensor.

Měření vzdálenosti objektu od senzoru v rozmezí 0–255 cm s přesností  $\pm 3$  cm. Senzor musí být připojen do portu označeného I2C, nelze připojit více senzorů typu NXT Ultrasonic Sensor.

**Inicializace:** robot.init\_sensor(sensor\_type=Sensor.NXT\_ULTRASONIC)

**Deinicializace:** robot.deinit\_sensor(sensor\_type=Sensor.NXT\_ULTRASONIC)

**Přístup:** robot.sensors.ultra\_nxt

```
distance()
```

Změří vzdálenost a navrátí ji.

**Vrací:** Vzdálenost objektu od senzoru.

**Typ:** int (0-255)

```
1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor
3
4 # Inicializace NXT ultrazvukového senzoru
5 robot.init_sensor(sensor_type=Sensor.NXT_ULTRASONIC)
6
7 # Změření vzdálenosti
8 distance = robot.sensors.ultra_nxt.distance()
```

Kód 12: Použití NXT ultrazvukového senzoru.

## 4.4 Zvukový senzor

```
class SoundSensor(port)
```

NXT Sound Sensor.

Měření intenzity okolního zvuku.

**Parametry:** `port` (`Port`) – Port, ke kterému je senzor připojen.  
**Inicializace:** `robot.init_sensor(sensor_type=Sensor.NXT_SOUND, port=Port)`  
**Deinicializace:** `robot.deinit_sensor(sensor_type=Sensor.NXT_SOUND, port=Port)`  
**Přístup:** `robot.sensors.sound[Port]`

```
intensity()
```

Změří intenzitu zvuku a navrátí ji.

**Vrací:** Intenzita zvuku. 0 pro největší intenzitu, 32768 pro nejmenší.  
**Typ:** `int` (0–32768)

```
1 # Importování konstant typů senzorů a portů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace NXT zvukového senzoru na senzorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.NXT_SOUND, port=Port.S1)
6
7 # Změření hladiny zvuku
8 sound_intensity = robot.sensors.sound[Port.S1].intensity()
```

Kód 13: Použití NXT zvukového senzoru.

## 5 EV3 senzory

### 5.1 Tlačítko

```
class TouchSensor(port)
```

EV3 Touch Sensor.

Informace o stavu stisknutí tlačítka.

**Parametry:** `port` (`Port`) – Port, ke kterému je senzor připojen.  
**Inicializace:** `robot.init_sensor(sensor_type=Sensor.EV3_TOUCH, port=Port)`.  
**Deinicializace:** `robot.deinit_sensor(sensor_type=Sensor.EV3_TOUCH, port=Port)`.  
**Přístup:** `robot.sensors.touch[Port]`.

```
pressed()
```

Navrátí stav tlačítka.

**Vrací:** `True` pokud je tlačítko stisknuté, `False` pokud není.  
**Typ:** `bool`

```

1 # Importování konstant typů senzorů a portů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace NXT tlačítka na sensorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.EV3_TOUCH, port=Port.S1)
6
7 # Zjištění stavu stisknutí tlačítka
8 touch_pressed = robot.sensors.touch[Port.S1].pressed()

```

Kód 14: Použití EV3 tlačítka.

## 5.2 Světelný senzor

*class* ColorSensor(*port*)

EV3 Color Sensor.

Měření světelné intenzity.

**Parametry:** **port** (*Port*) – Port, ke kterému je senzor připojen.  
**Inicializace:** `robot.init_sensor(sensor_type=Sensor.EV3_COLOR)`  
**Deinicializace:** `robot.deinit_sensor(sensor_type=Sensor.EV3_COLOR)`  
**Přístup:** `robot.sensors.light[Port]`

**reflection()**

Navrátí poslední změřenou odraženou intenzitu světla. Červená LED se střídavě zapíná a vypíná. Je změřena hodnota pro zapnutou LED, od které je odečtena hodnota pro vypnutou LED. Výsledná hodnota je upravena za použití kalibrace v senzoru a přemapována na interval 0-100 %.

**Vrací:** Odražená intenzita světla.  
**Typ:** int (0-100 %)

**reflection\_raw()**

Navrátí poslední změřenou odraženou intenzitu světla v neupraveném formátu. Červená LED se střídavě zapíná a vypíná. Je změřena hodnota pro zapnutou LED, od které je odečtena hodnota pro vypnutou LED.

**Vrací:** Odražená intenzita světla.  
**Typ:** int

**ambient()**

Navrátí poslední změřenou intenzitu okolního světla. Výsledná hodnota je upravena za použití kalibrace v senzoru a přemapována na interval 0-100 %.

**Vrací:** Intenzita okolního světla.  
**Typ:** int (0-100 %)

**rgb\_raw()**

Navrátí poslední změřenou odraženou intenzitu světla pro zapnutou červenou, modrou a zelenou LED.



**Vrací:** Tuple odražené intenzity světla (červená, modrá, zelená).  
**Typ:** (int, int, int)

#### rgb()

Navrátí poslední změřenou odraženou intenzitu světla pro zapnutou červenou, modrou a zelenou LED. Výsledná hodnota jsou upraveny za použití Open-Cube kalibrace a přemapovány na interval 0-100 %.

**Vrací:** Tuple odražené intenzity světla (červená, modrá, zelená).  
**Typ:** (int, int, int) (0-100 %)

#### hsv()

Navrátí hodnoty HSV pro poslední změřenou odraženou intenzitu světla při zapnuté červené, modré a zelené LED.

**Vrací:** Tuple HSV.  
**Typ:** (hue, saturation, value)

#### color()

Navrátí konstantu pro senzorem změřenou barvu. Lze použít konstanty barev `Color`.

**Vrací:** Konstanta změřené barvy.  
**Typ:** int (1-7 – black, blue, green, yellow, red, white, brown)

```
1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace EV3 světelného senzoru na sensorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.EV3_COLOR, port=Port.S1)
6
7 # Zjistění intenzity odraženého světla
8 reflection = robot.sensors.light[Port.S1].reflection()
```

Kód 15: Použití EV3 světelného senzoru.

## 5.3 Ultrazvukový dálkoměr

### `class UltrasonicSensor(port)`

EV3 Ultrasonic Sensor.

Měření vzdálenosti objektu od senzoru v rozmezí 0–2550 mm s přesností  $\pm 30$  mm.

**Parametry:** `port` (`Port`) – Port, ke kterému je senzor připojen.  
**Inicializace:** `robot.init_sensor(sensor_type=Sensor.EV3_ULTRA)`  
**Deinicializace:** `robot.deinit_sensor(sensor_type=Sensor.EV3_ULTRA)`  
**Přístup:** `robot.sensors.ultrasonic[port]`

#### distance()

Navrátí poslední změřenou vzdálenost.

**Vrací:** Vzdálenost objektu od senzoru.  
**Typ:** int (0-2550)

### presence()

Zjistí přítomnost jiného ultrazvukového senzoru.

**Vrací:** True pokud je přítomen jiný ultrazvukový senzor, jinak False.

**Typ:** bool

```
1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace EV3 ultrazvukového senzoru na sensorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.EV3_ULTRA, port=Port.S1)
6
7 # Změření vzdálenosti
8 distance = robot.sensors.ultrasonic[Port.S1].distance()
```

Kód 16: Použití EV3 ultrazvukového dálkoměru.

## 5.4 Infračervený senzor

### *class* InfraredSensor(*port*)

EV3 Infrared Sensor.

Měření vzdálenosti k nejbližšímu povrchu a ovládání dálkovým ovladačem.

**Parametry:** **port** (*Port*) – Port, ke kterému je senzor připojen.

**Inicializace:** robot.init\_sensor(sensor\_type=*Sensor.EV3\_INFRARED*)

**Deinicializace:** robot.deinit\_sensor(sensor\_type=*Sensor.EV3\_INFRARED*)

**Přístup:** robot.sensors.infrared[*Port*]

### distance()

Změří relativní vzdálenost k nejbližšímu povrchu. 100 % je přibližně 70 cm.

**Vrací:** Relativní vzdálenost povrchu od senzoru.

**Typ:** int (0-100 %)

### seeker(*channel*)

Zjistí přítomnost dálkového ovladače na daném kanálu.

**Parametry:** **channel** (int) – Nová hodnota, kterou zobrazí měření, pokud se senzor nepohne.

**Vrací:** Tuple směru (-25 je nejvíce vlevo, 25 nejvíce vpravo) a vzdálenosti (100 % je přibližně 200 cm, -128, pokud není ovladač nalezen) ovladače.

**Typ:** (int, int) (-25-25, 0-100 %)

```
1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace EV3 ultrazvukového senzoru na sensorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.EV3_INFRARED, port=Port.S1)
6
```

```

7 # Změření vzdálenosti
8 distance = robot.sensors.infrared[Port.S1].distance()

```

Kód 17: Použití EV3 infračerveného senzoru.

## 5.5 Gyroskop

```
class GyroSensor(port)
```

EV3 Gyro Sensor.

Měření úhlu a úhlové rychlosti v jedné ose.

**Parametry:** `port` (`Port`) – Port, ke kterému je senzor připojen.  
**Inicializace:** `robot.init_sensor(sensor_type=Sensor.EV3_GYRO)`  
**Deinicializace:** `robot.deinit_sensor(sensor_type=Sensor.EV3_GYRO)`  
**Přístup:** `robot.sensors.gyro[Port]`

**angle()**

Navrátí úhel natočení senzoru.

**Vrací:** Úhel natočení.  
**Typ:** int (°)

**speed()**

Navrátí úhlovou rychlost otáčení senzoru.

**Vrací:** Úhlová rychlost otáčení.  
**Typ:** int (°/s)

**reset\_angle(*angle*)**

Umožňuje vynulovat měření úhlu nebo nastavit novou počáteční hodnotu úhlu.

**Parametry:** `angle` (int) – Nová hodnota, kterou zobrazí měření, pokud se senzor nepohne.

**angle\_and\_speed()**

Navrátí úhel natočení a úhlovou rychlost otáčení senzoru.

**Vrací:** Tuple úhlu natočení a úhlové rychlosti otáčení.  
**Typ:** (int, int) (°, °/s)

**coarse\_speed()**

Navrátí úhlovou rychlost otáčení senzoru s nižším rozlišením a větším rozsahem.

**Vrací:** Úhlová rychlost otáčení.  
**Typ:** int (°/s)

```

1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace EV3 gyroskopu na senzorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.EV3_GYRO, port=Port.S1)
6
7 # Vynulování úhlu natočení
8 robot.sensors.gyro[Port.S1].reset_angle(0)
9
10 # Změření úhlu natočení a úhlové rychlosti
11 (angle, speed) = robot.sensors.gyro[Port.S1].angle_and_speed()

```

Kód 18: Použití EV3 gyroskopu.

## 6 Open-Cube senzory

### 6.1 Světelný senzor

```
class ColorSensor(port)
```

Open-Cube RGB optical reflective and color sensor.

Měření světelné intenzity.

**Parametry:** `port` (`Port`) – Port, ke kterému je senzor připojen.  
**Inicializace:** `robot.init_sensor(sensor_type=Sensor.OC_COLOR)`  
**Deinicializace:** `robot.deinit_sensor(sensor_type=Sensor.OC_COLOR)`  
**Přístup:** `robot.sensors.light[Port]`

```
reflection()
```

Navrátí poslední změřenou odraženou intenzitu světla. Červená LED se střídavě zapíná a vypíná. Je změřena hodnota pro zapnutou LED, od které je odečtena hodnota pro vypnutou LED. Výsledná hodnota je upravena za použití kalibrace v senzoru a přemapována na interval 0-100 %.

**Vrací:** Odražená intenzita světla.  
**Typ:** int (0-100 %)

```
reflection_raw()
```

Navrátí poslední změřenou odraženou intenzitu světla v neupraveném formátu. Červená LED se střídavě zapíná a vypíná. Je změřena hodnota pro zapnutou LED, od které je odečtena hodnota pro vypnutou LED.

**Vrací:** Odražená intenzita světla.  
**Typ:** int

```
reflection_raw_green()
```

Navrátí poslední změřenou odraženou intenzitu světla v neupraveném formátu. Zelená LED se střídavě zapíná a vypíná. Je změřena hodnota pro zapnutou LED, od které je odečtena hodnota pro vypnutou LED.

**Vrací:** Odražená intenzita světla.  
**Typ:** int

### reflection\_raw\_blue()

Navrátí poslední změřenou odraženou intenzitu světla v neupraveném formátu. Modrá LED se střídavě zapíná a vypíná. Je změřena hodnota pro zapnutou LED, od které je odečtena hodnota pro vypnutou LED.

**Vrací:** Odražená intenzita světla.  
**Typ:** int

### ambient()

Navrátí poslední změřenou intenzitu okolního světla. Výsledná hodnota je upravena za použití kalibrace v senzoru a přemapována na interval 0-100 %.

**Vrací:** Intenzita okolního světla.  
**Typ:** int (0-100 %)

### rgb\_raw()

Navrátí poslední změřenou odraženou intenzitu světla pro zapnutou červenou, modrou a zelenou LED.

**Vrací:** Tuple odražené intenzity světla (červená, modrá, zelená).  
**Typ:** (int, int, int)

### rgb()

Navrátí poslední změřenou odraženou intenzitu světla pro zapnutou červenou, modrou a zelenou LED. Výsledná hodnoty jsou upraveny za použití Open-Cube kalibrace a přemapovány na interval 0-100 %.

**Vrací:** Tuple odražené intenzity světla (červená, modrá, zelená).  
**Typ:** (int, int, int) (0-100 %)

### hsv()

Navrátí hodnoty HSV pro poslední změřenou odraženou intenzitu světla při zapnuté červené, modré a zelené LED.

**Vrací:** Tuple HSV.  
**Typ:** (hue, saturation, value)

### color()

Navrátí konstantu pro senzorem změřenou barvu. Lze použít konstanty barev `Color`.

**Vrací:** Konstanta změřené barvy.  
**Typ:** int (1-7 – black, blue, green, yellow, red, white, brown)

```
1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace Open-Cube světelného senzoru na sensorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.OC_COLOR, port=Port.S1)
```

```

6
7 # Zjistění intenzity odraženého světla
8 reflection = robot.sensors.light[Port.S1].reflection()

```

Kód 19: Použití Open-Cube světelného senzoru.

## 6.2 Laserový dálkoměr

```
class LaserSensor(port)
```

Open-Cube LIDAR distance sensor.

Měření vzdálenosti objektu od senzoru v rozmezí 0–4000 mm. Minimální zaručená vzdálenost měření je 4 cm. Pokud se objekt nachází blíže, než je tato vzdálenost, senzor změří nepřesnou hodnotu.

**Parametry:** `port` (`Port`) – Port, ke kterému je senzor připojen.  
**Inicializace:** `robot.init_sensor(sensor_type=Sensor.OC_LASER)`  
**Deinicializace:** `robot.deinit_sensor(sensor_type=Sensor.OC_LASER)`  
**Přístup:** `robot.sensors.ultrasonic[Port]`

```
distance()
```

Navrátí poslední změřenou vzdálenost.

**Vrací:** Vzdálenost objektu od senzoru.  
**Typ:** int (0-4000)

```
distance_fov()
```

Navrátí poslední změřenou vzdálenost. Tento režim používá větší zorné pole senzoru.

**Vrací:** Vzdálenost objektu od senzoru.  
**Typ:** int (0-4000)

```
distance_short()
```

Navrátí poslední změřenou vzdálenost. Tento režim je přesnější pro kratší vzdálenosti (do 1300 mm) a silné okolní světlo.

**Vrací:** Vzdálenost objektu od senzoru.  
**Typ:** int (0-4000)

```
set_led_distance(distance)
```

Nastaví mezní vzdálenost, při které se rozsvítí zelená LED senzoru.

**Parametry:** `distance` (int) – Mezní vzdálenost v mm.

```

1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace EV3 ultrazvukového senzoru na senzorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.OC_LASER, port=Port.S1)
6

```

```
7 # Změření vzdálenosti
8 distance = robot.sensors.laser[Port.S1].distance()
```

Kód 20: Použití Open-Cube laserového dálkoměru.

## 7 Servomotor

Motory ze sady NXT a EV3 jsou vnitřními parametry identické, liší se pouze vzhledem a plastovou konstrukcí.

```
class Motor(port)
```

Na motorech se nachází rotační enkodéry umožňující snímání polohy natočení motoru a aproximaci rychlosti otáčení.

Poskytuje tři módy řízení – výkon, poloha, rychlost.

V módu řízení výkonu uživatel přímo nastavuje požadovaný výkon na motoru v rozmezí -100–100 %.

V módu řízení polohy uživatel volí požadovanou polohu natočení motoru ve °. Regulace je zprostředkována PID regulátorem s nastavitelnými konstantami.

V módu řízení rychlosti uživatel volí požadovanou rychlost otáčení motoru ve °/s. Maximální rychlost při nezatíženém motoru je přibližně 950 °/s. Regulace je zprostředkována PID regulátorem s nastavitelnými konstantami.

**Parametry:** **port** (**Port**) – Port, ke kterému je motor připojen.

**Inicializace:** `robot.init_motor(port=Port)`

**Deinicializace:** `robot.deinit_motor(port=Port)`

**Přístup:** `robot.motor[Port]`

**Parametry:** **port** (**Port**) – Port, ke kterému je motor připojen.

```
set_power(power)
```

Nastaví výkon na motoru.

**Parametry:** **power** (float) – Požadovaný výkon na motoru v rozmezí -100–100 %. Při zadání hodnoty mimo toto rozmezí je nastavena nejbližší krajní hodnota.

```
init_encoder()
```

Inicializuje enkodér s měřením polohy a rychlosti motoru.

```
deinit_encoder()
```

Denicializuje enkodér s měřením polohy a rychlosti motoru.

```
position()
```

Navrátí polohu motoru.

**Vrací:** Poloha motoru.  
**Typ:** úhel: °

### **speed()**

Navrátí polohu motoru.

**Vrací:** Rychlost motoru.  
**Typ:** úhlová rychlost: °/s

### **init\_regulator()**

Inicializuje regulátor motoru.

### **deinit\_regulator()**

Deinicializuje regulátor motoru.

### **set\_regulator\_position(*position*)**

Nastaví motor do módu řízení polohy.

**Parametry:** **position** (int) – Požadovaná konečná poloha motoru ve °.

### **regulator\_pos\_set\_consts(*p, i, d*)**

Nastaví PID konstanty regulátoru polohy. Výchozí hodnoty jsou 0.9, 0.1 a 0.

**Parametry:**

- **p** (float) – Konstanta proporcionální složky.
- **i** (float) – Konstanta integrační složky.
- **d** (float) – Konstanta derivační složky.

### **set\_regulator\_speed(*speed*)**

Nastaví motor do módu řízení polohy.

**Parametry:** **speed** (int) – Požadovaná konečná rychlost motoru ve °/s.

### **regulator\_speed\_set\_consts(*p, i, d*)**

Nastaví PID konstanty regulátoru rychlosti. Výchozí hodnoty jsou 0.1, 0.8 a 0.

**Parametry:**

- **p** (float) – Konstanta proporcionální složky.
- **i** (float) – Konstanta integrační složky.
- **d** (float) – Konstanta derivační složky.

### **regulator\_error()**

Navrátí aktuální regulační odchylku.



- Vrací:** Regulační odchylka.
- Typ:**
- **mód řízení polohy** – poloha: °
  - **mód řízení rychlosti** – úhlová rychlost: °/s

### **regulator\_power()**

Navrátí aktuální akční zásah regulátoru.

- Vrací:** Akční zásah.
- Typ:** výkon (-100–100 %)

```

1 # Importování konstant portů
2 from lib.robot_consts import Port
3
4 # Inicializace motorů na motorových portech 1 a 2
5 robot.init_motor(Port.M1)
6 robot.init_motor(Port.M2)
7
8 # Nastavení výkonu motorů na 50 % a 20 % v opačném směru otáčení
9 robot.motors[Port.M1].set_power(50)
10 robot.motors[Port.M2].set_power(-20)
11
12 # Inicializace enkodérů pro měření pozice a rychlosti motorů
13 robot.motors[Port.M1].init_encoder()
14 robot.motors[Port.M2].init_encoder()
15
16 # Zjištění aktuální pozice a rychlosti motorů
17 pos1 = robot.motors[Port.M1].position()
18 pos2 = robot.motors[Port.M2].position()
19 speed1 = robot.motors[Port.M1].speed()
20 speed2 = robot.motors[Port.M2].speed()

```

Kód 21: Použití motorů.

## 8 Parametry a konstanty

Parametry a konstanty importovatelné pomocí `lib.robot_consts`.

### 8.1 Sensor

#### **class** Sensor

Typy senzorů.

#### **NO\_SENSOR**

Žádný senzor.

#### **NXT\_LIGHT**

NXT světelný senzor.

#### **NXT\_ULTRA**

NXT ultrazvukový dálkoměr.

### **NXT\_TOUCH**

NXT tlačítko.

### **NXT\_SOUND**

NXT zvukový senzor.

### **GYRO\_ACC**

ICM20608-G gyroskop a akcelerometr.

### **EV3\_COLOR**

EV3 světelný senzor.

### **EV3\_GYRO**

EV3 gyroskop.

### **EV3\_INFRARED**

EV3 infračervený senzor.

### **EV3\_TOUCH**

EV3 tlačítko.

### **EV3\_ULTRASONIC**

EV3 ultrazvukový dálkoměr.

### **OC\_LASER**

Open-Cube laserový dálkoměr.

### **OC\_COLOR**

Open-Cube světelný senzor.

## 8.2 Port

### *class* Port

Porty motorů a senzorů na kostce.

### **M1**

Port motoru 1.

### **M2**

Port motoru 2.

### **M3**

Port motoru 3.

**M4**

Port motoru 4.

**S1**

Port senzoru 1.

**S2**

Port senzoru 2.

**S3**

Port senzoru 3.

**S4**

Port senzoru 4.

### 8.3 Button

*class* **Button**

Tlačítka na kostce.

**POWER**

Tlačítko zapnout.

**LEFT**

Tlačítko doleva.

**RIGHT**

Tlačítko doprava.

**OK**

Tlačítko OK.

**UP**

Tlačítko nahoru.

**DOWN**

Tlačítko dolů.

### 8.4 Light

*class* **Light**

Změřená hodnota při vypnuté a zapnuté LED NXT světelného senzoru.

**OFF**

Změřená hodnota při vypnuté LED.

**ON**

Změřená hodnota při zapnuté LED.

## 8.5 GyroAcc

***class* GyroAcc**

Změřená hodnota vestavěného gyroskopu a akcelerometru ICM42688 v osách x, y, z a pin přerušení senzoru.

**AX**

Zrychlení z akcelerometru v ose x.

**AY**

Zrychlení z akcelerometru v ose y.

**AZ**

Zrychlení z akcelerometru v ose z.

**GX**

Úhlová rychlost z gyroskopu v ose x.

**GY**

Úhlová rychlost z gyroskopu v ose y.

**GZ**

Úhlová rychlost z gyroskopu v ose z.

**IRQ\_PIN**

Pin přerušení senzoru ICM20608-G.

## 8.6 Color

***class* Color**

Barva detekovaná EV3 a Open-Cube světelnými senzory.

**BLACK**

**BLUE**

**GREEN**

**YELLOW**

RED

WHITE

BROWN

## 9 Užitečné funkce MicroPythonu

### 9.1 Výpis informací

```
1 x, y = 64, 128.4096
2 y_string = "y"
3
4 # 1. varianta - oddělení čárkami
5 # Tato varianta je vhodná na jednoduché a rychlé zobrazení textu
6 # Jednotlivými prvky jsou odděleny mezerou
7 print("x =", x, ",", y_string, "=", y)
8 # x = 64, y = 128.4096
9
10 # 2. varianta - použití f-stringu
11 # Tato varianta umožňuje formátovat float čísla
12 # d označuje typ integer, s typ string a f typ float
13 # .2 určuje počet zobrazených desetinných míst
14 print(f"x = {x:d}, {y_string:s} = {y:.2f}")
15 # x = 64, y = 128.41
16
17 # Použití f-stringu je vhodné i pro zobrazení na displeji kostky
18 robot.display.fill(0)
19 # první řádek displeje
20 robot.display.text(f"x = {x:d}", 0, 0, 1)
21 # druhý řádek displeje
22 robot.display.text(f"{y_string:s} = {y:.2f}", 0, 10, 1)
23 robot.display.show()
24
25 # Výpis bez odřádkování
26 for i in range(10)
27     print(i, end="")
28 print() # odřádkování na konci
29 # 123456789
```

Kód 22: Výpis informací.

### 9.2 Generování náhodných čísel

Ukázka generování náhodných čísel.

- [MicroPython dokumentace knihovny random](https://docs.micropython.org/en/latest/library/random.html) [ <https://docs.micropython.org/en/latest/library/random.html> ]

```
1 # Importování funkcí knihovny random
2 from random import random, randint
3
4 random() # Náhodné číslo typu float v intervalu [0.0, 1.0)
5
```

```

6 a, b = 0, 100
7 randint(a, b) # Náhodné číslo typu int (celé číslo) v intervalu [a, b]

```

Kód 23: Použití knihovny random.

## 9.3 Čas

Ukázka funkcí pro uspání programu, měření času a periodického volání funkce pomocí timeru.

- [MicroPython dokumentace knihovny time](https://docs.micropython.org/en/latest/library/time.html) [ <https://docs.micropython.org/en/latest/library/time.html> ]
- [MicroPython dokumentace třídy Timer](https://docs.micropython.org/en/latest/library/machine.Timer.html) [ <https://docs.micropython.org/en/latest/library/machine.Timer.html> ]

```

1 # Importování funkcí knihovny sleep
2 from time import sleep, sleep_ms, sleep_us, ticks_us, ticks_diff
3
4 start_time = ticks_us() # Zjištění počátečního času v mikrosekundách
5
6 sleep(1) # Uspání programu na 1 sekundu
7 sleep_ms(2) # Uspání programu na 2 milisekundy
8 sleep_us(3) # Uspání programu na 3 mikrosekundy
9
10 end_time = ticks_us() # Zjištění koncového času v mikrosekundách
11 print(ticks_diff(end_time, start_time), "us") # 1002003 us

```

Kód 24: Použití knihovny time.

```

1 # Importování třídy Timer
2 from machine import Timer
3
4 counter = 0
5
6 # Definice callback funkce pro timery
7 def my_callback(t):
8     global counter
9     counter += 1
10    print(counter)
11
12 # Inicializace timeru, který volá callback funkci periodicky
13 # s frekvencí 100 Hz
14 # -1 značí virtuální timer (mikrokontrolér rp2040 neumožňuje použití
15 # hardwarových timerů)
16 timer1 = Timer(-1)
17 timer1.init(mode=Timer.PERIODIC, freq=100, callback=my_callback)
18
19 # Inicializace timeru, který zavolá callback funkci pouze jednou po
20 # uplynutí 1000 ms
21 timer2 = Timer(-1)
22 timer2.init(mode=Timer.ONE_SHOT, period=1000, callback=my_callback)
23
24 # Vypnutí periodického timeru
25 timer1.deinit()

```

Kód 25: Použití třídy Timer.

## 9.4 Manipulace s binárními daty

Tyto funkce jsou užitečné při práci s binárními daty např. při bezdrátové komunikaci přes ESP32.

- [Micropython dokumentace knihovny struct](https://docs.micropython.org/en/latest/library/struct.html) [ <https://docs.micropython.org/en/latest/library/struct.html> ]

```
1 # Importování knihovny struct
2 import struct
3
4 # Vytvoření binárních dat ze tří čísel. Formát čísel je určen stringem
  prvního argumentu. Formát stringu je podrobně popsán v dokumentaci
  knihovny struct.
5 # V tomto případě je první číslo uloženo jako typ int (4 byty), druhé
  jako float (4 byty) a třetí jako double (8 bytů)
6 binary_data = struct.pack("@ifd", 12, 1.234, 5)
7
8 # Konverze binárních dat
9 values = struct.unpack("@ifd", binary_data)
10
11 # Spočítá velikost paměti v bytech potřebnou k uložení binárních dat
12 byte_size = struct.calcsize("@ifd")
13 print(byte_size) # 16
```

Kód 26: Použití knihovny struct.

## 9.5 Vícejádrové programování

Mikrokontrolér rp2040 má 2 jádra, defaultně se používá pouze jedno jádro. Použití druhého jádra je možné, ale aktuálně je v MicroPythonu pouze experimentální.

- [Python dokumentace knihovny \\_thread](https://docs.python.org/3.5/library/_thread.html) [ [https://docs.python.org/3.5/library/\\_thread.html](https://docs.python.org/3.5/library/_thread.html) ]

```
1 # Importování knihovny _thread a funkce sleep
2 import _thread
3 from time import sleep
4
5 lock = _thread.allocate_lock()
6 counter = 0
7
8 # Definice funkce druhého jádra
9 def core2():
10     global counter
11     lock.acquire() # Uzamčení pro práci se společnými daty
12     counter += 1
13     lock.release() # Uvolnění
14     _thread.exit() # Ukončení běhu jádra
15
16 # Aktivace druhého jádra
17 _thread.start_new_thread(core2, ())
18
19 lock.acquire() # Uzamčení pro práci se společnými daty
20 print(counter)
21 lock.release() # Uvolnění
```

Kód 27: Použití knihovny \_thread.

## 10 Použité pojmy Pythonu

Datové typy:

- **int** (integer) – celé číslo (12, -51),
- **float** – desetinné číslo (120.32, -0.1248),
- **bool** – pravdivostní hodnota (True, False),
- **str** (string) – textový řetězec ("abc", "Open-Cube").

Datové struktury:

- **list** – posloupnost dat, která může obsahovat prvky různých datových typů. Jednotlivé prvky lze měnit, přidávat a odebírat. Zapisuje se do hranatých závorek a prvky jsou odděleny čárkou. [1, "Open-Cube", True, 0.23]
- **tuple** – posloupnost dat, která může obsahovat prvky různých datových typů. Jednotlivé prvky nelze měnit ani přidávat a odebírat. Zapisuje se do kulatých závorek a prvky jsou odděleny čárkou. (1, "Open-Cube", True, 0.23)

Klíčová slova:

- **None** – žádná hodnota (senzor nefunguje, nedokázal změřit hodnotu atd.).