

Uživatelský manuál programovacího jazyka Python pro LEGO MINDSTORMS EV3

ADAM JÁNEŠ

Prvotní informace

- Využíváme MicroPython

- První řádek musí vypadat následovně (kvůli překladu MicroPythonu):
`#!/usr/bin/env pybricks-micropython`

- Využíváme třídy a funkce knihovny Pybricks

- Pro správnou funkci importujeme oddíly knihovny Pybricks

```
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                InfraredSensor, UltrasonicSensor, GyroSensor)
from pybricks.parameters import Port, Stop, Direction, Button, Color
from pybricks.tools import wait, StopWatch, DataLog
from pybricks.robotics import DriveBase
from pybricks.media.ev3dev import SoundFile, ImageFile, Font
from pybricks.messaging import BluetoothMailboxServer, TextMailbox, BluetoothMailboxClient,
                             NumericMailbox
```

Rozcestník

Tlačítka

Reproduktor

Obrazovka

Obrazové
stopy

Čas

Baterie

Parametry
a
konstanty

Motor

Bluetooth
komunikace

Záznam
dat

Komplexní
funkce pro
pohyb
robota

Senzory

Barvový
senzor

Slučovací
senzor

Gyroskopický
senzor

Ultrazvukový
senzor



**Parametry
a
konstanty**

Parametry a konstanty

- Pro určení portů ([typ Port](#))
- Pro určení kladného směru rotace ([typ Direction](#))
- Pro chování servomotoru po zastavení ([typ Stop](#))
- Pro určení barev ([typ Color](#))
- Pro určení tlačítek na EV3 ([typ Button](#))
- Pro přístup k obrázkům z knihovny ImageFile ([typ Image](#))
- Pro přístup k zvukovým stopám z knihovny SoundFile ([typ Sound](#))
- Pro vytvoření stylu písma ([typ Font](#))

Class Port

- Rozlišuje, na kterém portu je periférie zapojena
- Pro motory {A, B, C, D}
- Pro senzory {S1, S2, S3, S4}

```
# inicializace motoru zapojeného do portu B  
mot=Motor(Port.B)  
  
# inicializace světelného senzoru na port 2  
senzorL = ColorSensor(Port.S2)
```

Class Direction

- Určuje kladný směr otáčení servomotoru
- Po směru hodinových ručiček
CLOCKWISE
- Proti směru hodinových ručiček
COUNTERCLOCKWISE

```
mot=Motor(Port.B, Direction.COUNTERCLOCKWISE)
```

```
mot=Motor(Port.C, Direction.CLOCKWISE)
```

Class Stop

- Určuje chování motoru po zastavení
- Kola lze volně otáčet
COAST
- Pasivně brzdí kola
BRAKE
- Udržuje motor ve stejném úhlu
HOLD

```
mot=Motor(Port.B)
```

```
mot.run(100) #toci rychlosti 100deg/s, dokud neskončí  
kód nebo neproběhne příkaz pracující se servomotorem
```

```
wait(100) #čeká 100 ms
```

```
#zastaví motor po otočení o 360 ° , ale následně je  
možné otáčet volně servomotorem
```

```
mot.run_angle(100, 360, then = Stop.COAST, wait= True)
```

```
#zastaví motor a následně pasivně zabraňuje rotaci  
servomotoru
```

```
mot.run_angle(100, 360, then = Stop.BRAKE, wait= True)
```

```
#zastaví motor a zamezuje vychýlení od cíleného  
úhlu
```

```
mot.run_angle(100, 360, then = Stop.HOLD, wait= True)
```

```
wait(100)
```


Class Color

- Definice z 9 předdefinovaných barev
- Předdefinované barvy {černá, modrá, zelená, žlutá, červená, bílá, oranžová, fialová}
- Jejich pojmenování v kódu {BLACK, BLUE, GREEN, YELLOW, RED, WHITE, ORANGE, PURPLE}

#ulozeni definovanych barev do promennych

```
cerna=Color.BLACK
```

```
modra=Color.BLUE
```

```
zelená=Color.GREEN
```

```
zluta=Color.YELLOW
```

```
cervena=Color.RED
```

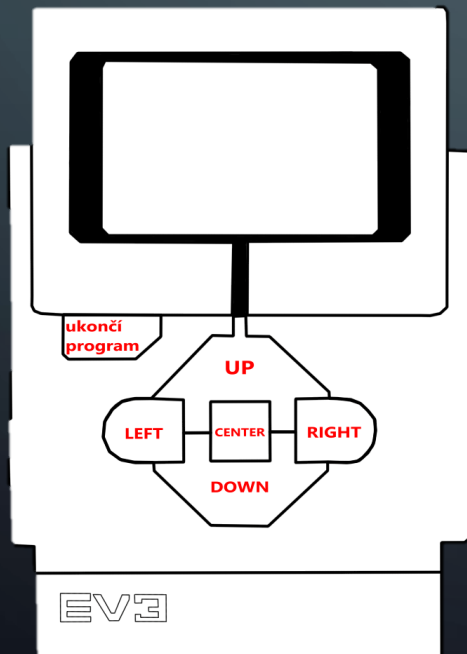
```
bila=Color.WHITE
```

```
oranzova=Color.ORANGE
```

```
fialova=Color.PURPLE
```

Class Button

- Definice tlačítek na EV3
- Definovaná tlačítka
{UP, LEFT, CENTER, RIGHT,
DOWN}



```
ev3 = EV3Brick()
```

```
#vraci pole zmacknutych tlačítek  
tlacitka=ev3.buttons.pressed()  
#pokud na pozici 0 je prostřední  
tlačítka, vypíše celé pole
```

```
if tlacitka[0] ==Button.CENTER:  
    print(tlacitka)
```

Class Font(family=None, size=12, bold=False, monospace=False, lang=None, script=None)

- Vrací font, který je nejbližší vstupním parametrům
- Family – (str) určuje styl písma
- Size – (int) určuje velikost písma <6,24>
- Bold – (bool) určuje, zda se preferuje tučné písmo
- Monospace – (bool), určuje, zda každý znak má mít konstantní šířku
- Lang – (str) určuje jazykovou sadu
- Script –(str) určuje použité znakové sady (azbuka, latinka)

```
ev3 = EV3Brick()
```

```
big_font = Font(family='Times', size=24,  
bold=True, monospace=True, lang='cs')
```

Získání informací o vytvořeném fontu

- Family – vrací font
- Style – informace o tučnosti písma
- Width – vrací maximální šířku znaku
- Height – vrací výšku znaku
- Text_width – vrací šířku textu
- Text_height – vrací výšku textu

```
ev3 = EV3Brick()

big_font = Font(family='Times',size=24, bold=True,
monospace=True, lang='cs ')
print(big_font.family) #vraci font
# vraci jestli je tucne nebo klasicke
print(big_font.style)
print(big_font.width) # maximalni tloustka znaku
print(big_font.height) # vyska znaku
print(big_font.text_width('a')) #sirka textu
# sirka textu, dvojnásobna narozdil od
predchoziho radku
print(big_font.text_width('aa'))
print(big_font.text_height('a')) # vyska textu
# vyska textu, stejna jako v predchozim radku
print(big_font.text_height('abfvdšdvndvedaa'))
```

Class ImageFile

- Obsahuje sadu obrázků

```
ev3 = EV3Brick()
```

```
obr=ImageFile.BACKWARD
```

```
ev3.screen.draw_image(0, 0, obr)
```

```
wait(3000)
```

Seznam dostupných obrázků

- Sada Information
{ACCEPT, BACKWARD, DECLINE, FORWARD, LEFT, NO_GO, QUSETION_MARK, RIGHT, STOP_1, STOP_2, THUMBS_DOWN, THUMBS_UP, WARNING}
- Sada LEGO
{EV3, EV3_ICON}
- Sada Objects
{TARGET}
- Sada Eyes
{ANGRY, AWAKE, BOTTOM_LEFT, BOTTOM_RIGHT, CRAZY_1, CRAZY_2, DIZZY, DOWN, EVIL, KNOCKED_OUT, MIDDLE_LEFT, MIDDLE_RIGHT, NEUTRAL, PINCHED_LEFT, PINCHED_RIGHT, SLEEPING, TIRED_LEFT, TIRED_MIDDLE, TIRRED_RIGHT, UP, WINKING}

Class SoundFile

- Obsahuje sadu zvukových stop

```
ev3 = EV3Brick()  
zvuk=SoundFile.ERROR  
#prehraje dany soubor  
ev3.speaker.play_file(zvuk)
```

Seznam dostupných zvukových stop

- Sada Expression
{BOING, BOO, CHEERING, CRUNCHING, CRYING, FANFARE, LAUGHING_1, LAUGHING_2, MAGIC_WAND, OUCH, SHOUTING, SNEEZING, UH_OH}
- Sada Information
{ACTIVATE, ANALYZE, BACKWARDS, COLOR, DETECTED, DOWN, ERROR, ERROR_ALARM, FLASHING, FORWARD, LEFT, OBJECT, RIGHT, SEARCHING, START, STOP, TOUCH, TURN, UP}
- Sada Communication
{BRAVO, EV3, FANTASTIC, GAME_OVER, GO, GOOD_JOB, GOOD, GOODBYE, HELLO, HI LEGO, MINDSTORMS, MORNING, NO, OKAY, OKEY_DOKEY, SORRY, THANK_YOU, YES}
- Sada Movement
{SPEED_DOWN, SPEED_IDLE, SPEED_UP}
- Sada Colors
{BLACK, BLUE, BROWN, GREEN, RED, WHITE, YELLOW}
- Sada Mechanical
{AIR_RELEASE, AIRBRAKE, BACKING_ALERT, HORN_1, HORN_2, LASER, MOTOR_IDLE, MOTOR_START, MOTOR_STOP, RATCHET, SONAR, TICK_TACK}
- Sada Animal
{CAT_PURR, DOG_BARK_1, DOG_BARK_2, DOG_GROWL, DOG_SNIFF, DOG_WHINE, ELEPHANT_CALL, INSECT_BEZZ_1, INSECT_BUZZ_2, INSECT_CHIRP, SNAKE_HISS, SNAKE_RATTLE, T_REX_ROAR}
- Sada Numbers
{ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN}
- Sada System
{CLICK, CONFIRM, GENERAL_ALERT, OVERPOWER, READY}



Čas

Určení času

- Pozastavení programu ([wait\(\)](#))
- Určení času ([time\(\)](#))
- Pozastavení času ([pause\(\)](#))
- Pokračování v měření času ([resume\(\)](#))
- Vynulování času ([reset\(\)](#))

Function wait(time)

- Time – (int) jak dlouho má být pozastaven běh programu [ms]

```
mot=Motor(Port.B)
#toci rychlosti 100deg/s, dokud
neskončí kód nebo neproběhne
příkaz pracující se servomotorem

mot.run(100)
#pozastavení průběhu programu na
100 ms

wait(100)
```

Function time()

- Součást třídy Stopwatch()
- Vrátí čas od inicializace třídy Stopwatch() nebo od posledního zavolání funkce reset() [ms]

```
#inicializace stopek (od této chvíle běží čas)
```

```
s=StopWatch()
```

```
wait(100)
```

```
#vypise současný čas ,tedy kolem 100
```

```
print(s.time() )
```

Function pause()

- Součást třídy Stopwatch()
- Pozastaví měření času třídou Stopwatch()

```
#inicializace stopek (od této chvíle běží čas)
```

```
s=StopWatch()  
wait(100)
```

```
#vypíše současný čas ,tedy kolem 100
```

```
print(s.time())
```

```
s.pause() #pozastaví měření času
```

```
wait(50) # tento čas se nezapočítává
```

```
# vypíše stejný nebo velmi blízký čas k  
prvnímu výpisu
```

```
print(s.time())
```

Function resume()

- Součást třídy Stopwatch()
- Zajistí pokračování měření času třídou Stopwatch()

```
#inicializace stopek (od této chvíle běží čas)
s=StopWatch()
wait(100)
#vypíše současný čas ,tedy kolem 100
print(s.time() )
s.pause() #pozastaví měření času
wait(50) # tento čas se nezapočítává
s.resume() #pokračování měření času
wait(30)
# vypíše současný čas ,tedy kolem 130
print(s.time())
```

Function reset()

- Součást třídy Stopwatch()
- Vynuluje čas třídy Stopwatch()
- Pokud je měření pozastaveno, čas zůstane pozastaven na 0

```
s=StopWatch()  
wait(100)  
#vypíše současný čas ,tedy kolem 100  
  
print(s.time())  
s.reset() #vynuluje měření času  
wait(20)  
# vypíše současný čas od posledního  
vynulování ,tedy kolem 20  
print(s.time())
```



Tlačítka

EV3 - tlačítko

- Pro zjištění, které tlačítka jsou zmáčknutá ([button.pressed\(\)](#))
- Zapnutí podsvícení tlačítek ([light.on\(\)](#))
- Vypnutí podsvícení tlačítek ([light.off\(\)](#))

Function `button.pressed()`

- Vrací seznam (list of Button) zmáčknutých tlačítek

```
ev3 = EV3Brick()  
tlacitka=ev3.buttons.pressed()  
print(tlacitka)
```

Function `light.on(Color)`

- Zapne podsvícení tlačítek
- `Color` –(`Color`) určuje barvu podsvícení tlačítek
- Dostupné barvy
{zelená, žlutá, oranžová, červená}
- Při zadání jiných barev je podsvícení vypnuto

```
ev3 = EV3Brick()  
ev3.light.on(Color.RED)  
wait(5000)
```

Function light.off()

- Vypne podsvícení tlačítek

```
ev3 = EV3Brick()  
ev3.light.off()  
wait(5000)
```



Reproduktor

EV3 - reproduktor

- Zazní zvuk o dané délce a frekvenci ([speaker.beep\(\)](#))
- Zazní dané tóny v daném tempu ([speaker.play_notes\(\)](#))
- Přehraje zvukový soubor ([speaker.play_file\(\)](#))
- Přehraje zadaný text ([speaker.say\(\)](#))
- Nastaví vlastnosti pro čtení textu ([speaker.set_speech_options\(\)](#))
- Nastaví hlasitost reproduktoru ([speaker.set_volume\(\)](#))

Function `speaker.beep(frequency=500, duration=100)`

- Zazní tón o dané frekvenci a délce `ev3 = EV3Brick()`
- `Frequency` – (int) určuje frekvenci [Hz] `ev3.speaker.beep(800, 1500)`
- `Duration` – (int) určuje délku trvání zvuku [ms]

Function `speaker.play_tones(tones, tempo=120)`

- Zahraje sekvenci not v daném tempu
- Notes – (array of sting) určuje noty a jejich délky
- Tempo – (int) určuje počet čtvrtových dob za minutu

```
ev3 = EV3Brick()
noty=['C4/4_', 'D4/4', 'E4/4_', 'F4/4',
      'G4/4_', 'A4/4', 'B4/4_', 'C5/4']
ev3.speaker.play_notes(noty, 150)
```


Vysvětlení znaků prvku pole

	Tón	Posuvka	Tónina	Délka tónu	Prodloužení délky tónu	Legato
Povinný	Ano	Ne	Ano	Ano	Ne	Ne
Význam	Určuje název noty	Mění výšku tónu o půltón	Určuje oktávu	Určuje délku noty	Možnost prodloužení noty o 1.5x	Naváže notu s následující notou
Seznam použitých znaků	{C, D, E, F, G, A, B}	{#, b}	{2, 3, 4, 5, 6, 7, 8}	{/1, /2, /4, /8, /16}	{.}	{_}

Tab 1:

Function `speaker.play_file(file_name)`

- Přehraje zadaný zvukový soubor
- `file_name` – (str) určuje název soubor i s příponou

```
ev3 = EV3Brick()  
#prehraje dany soubor  
ev3.speaker.play_file('boing.wav')
```

Function `speaker.say(text)`

- Přečte zadaný text
- Text – (str) označuje text, který bude přečten
- Jakým způsobem bude text přečten upravuje funkce `speaker.set_speech_options()`

```
ev3 = EV3Brick()  
#precte dany text  
ev3.speaker.say(' I love EV3' )
```

Function `speaker.set_speech_options(language=None, voice=None, speed=None, pitch=None)`

- Nastavuje vlastnosti, jak bude text přečten
- `Language` – (str) označuje jazyk, kterým bude text přečten
- `Voice` – (str) označuje hlas, jakým bude text přečten
- `Speed` – (int) určuje počet přečtených slov za minutu
- `Pitch` – (int) určuje výšku hlasu v rozsahu 0 až 99

```
ev3 = EV3Brick()
```

```
ev3.speaker.set_speech_options('cs',  
'f4', 150, 90)
```

```
ev3.speaker.say('Ahoj, jak se máš?')
```

Možné nastavení parametrů

- Dostupné jazyky

{'af': Afrikaans, 'an': Aragonese, 'bg': Bulgarian, 'bs': Bosnian, 'ca': Catalan, 'cs': Czech, 'cy': Welsh, 'da': Danish, 'de': German, 'el': Greek, 'en': English (default), 'en-gb': English (United Kingdom), 'en-sc': English (Scotland), 'en-uk-north': English (United Kingdom, Northern), 'en-uk-rp': English (United Kingdom, Received Pronunciation), 'en-uk-wmids': English (United Kingdom, West Midlands), 'en-us': English (United States), 'en-wi': English (West Indies), 'eo': Esperanto, 'es': Spanish, 'es-la': Spanish (Latin America), 'et': Estonian, 'fa': Persian, 'fa-pin': Persian, 'fi': Finnish, 'fr-be': French (Belgium), 'fr-fr': French (France), 'ga': Irish, 'grc': Greek, 'hi': Hindi, 'hr': Croatian, 'hu': Hungarian, 'hy': Armenian, 'hy-west': Armenian (Western), 'id': Indonesian, 'is': Icelandic, 'it': Italian, 'jbo': Lojban, 'ka': Georgian, 'kn': Kannada, 'ku': Kurdish, 'la': Latin, 'lfn': Lingua Franca Nova, 'lt': Lithuanian, 'lv': Latvian, 'mk': Macedonian, 'ml': Malayalam, 'ms': Malay, 'ne': Nepali, 'nl': Dutch, 'no': Norwegian, 'pa': Punjabi, 'pl': Polish, 'pt-br': Portuguese (Brazil), 'pt-pt': Portuguese (Portugal), 'ro': Romanian, 'ru': Russian, 'sk': Slovak, 'sq': Albanian, 'sr': Serbian, 'sv': Swedish, 'sw': Swahili, 'ta': Tamil, 'tr': Turkish, 'vi': Vietnamese, 'vi-hue': Vietnamese (Hue), 'vi-sgn': Vietnamese (Saigon), 'zh': Mandarin Chinese, 'zh-yue': Cantonese Chinese}

- Dostupné hlasy

{'f1': female variant 1, 'f2': female variant 2, 'f3': female variant 3, 'f4': female variant 4, 'f5': female variant 5, 'm1': male variant 1, 'm2': male variant 2, 'm3': male variant 3, 'm4': male variant 4, 'm5': male variant 5, 'm6': male variant 6, 'm7': male variant 7, 'croak': croak, 'whisper': whisper, 'whisperf': female whisper}

Function `speaker.set_volume(volume, which='_all_')`

- Změní hlasitost reproduktoru
- Volume – (int) určuje hlasitost reproduktoru [%]
- Which – (str) pro jaké funkce změna hlasitosti platí
 - 'Beep' – pro funkci `beep()`
 - 'PCM' – pro funkce `play_file()` a `say()`
 - '_all_' – pro obě výše zmíněné skupiny funkcí

```
ev3 = EV3Brick()  
ev3.speaker.set_volume(20,  
which='_all_')  
ev3.speaker.say('I love EV3')
```



Образовка

EV3 - obrazovka

- Psaní textu na obrazovku ([screen.draw_text\(\)](#), [screen.print\(\)](#))
- Změna stylu písma ([screen.set_font\(\)](#))
- Vymazání celé obrazovky ([screen.clear\(\)](#))
- Zobrazení obrázku ze souboru ([screen.load_image\(\)](#), [screen.draw_image\(\)](#))
- Zakreslení pixelu ([screen.draw_pixel\(\)](#))
- Zakreslení geometrických útvarů ([screen.draw_line\(\)](#), [screen.draw_box\(\)](#), [screen.draw_circle\(\)](#))
- Zjištění rozměrů obrazovky ([screen.height](#), [screen.width](#))
- Pořízení screenshotu obrazovky ([screen.save\(\)](#))

Function `screen.draw_text(x, y, text, text_color=Color.BLACK, background_color=None)`

- Zakreslí text na danou pozici
- X – (int) určuje horizontální souřadnici umístění textu
- Y – (int) určuje vertikální souřadnici umístění textu
- Text – (str) určuje text, který má být zakreslen
- Text_color – (Color) určuje barvu textu
- Background_color – (Color) určuje barvu pozadí

```
ev3 = EV3Brick()
ev3.screen.draw_text(0, 50,
                    "ahoj", Color.WHITE, Color.BLACK)
wait(2000)
```

Function `screen.print(*args, sep=' ', end='\n')`

- Vypíše text na obrazovku s bílým pozadím
- `*args` – (object) určuje objekty, jejichž hodnoty se mají vypsát
- `Sep` – (str) určuje znaky, které mají rozdělovat 2 objekty
- `End` – (str) určuje znaky, které budou vypsány za posledním objektem

```
ev3 = EV3Brick()
#ukazka scrollovani vypisu
for i in range(10):
    ev3.screen.print("ahoj")

wait(500)
#tento text nebude zobrazen cely
ev3.screen.print("ahoj, jak se mas? Co
delas?")
wait(2000)
```

Function `screen.set_font(font)`

- Změní styl písma
- Font – (Font) určuje proměnnou obsahující vlastnosti písma

```
ev3 = EV3Brick()
#nastaveni fontu
tiny_font = Font(size=6)
big_font = Font(size=24, bold=True)
ev3.screen.print('Hello!')
ev3.screen.set_font(tiny_font)
ev3.screen.print('hello')
ev3.screen.set_font(big_font)
ev3.screen.print('HELLO')
wait(5000)
```

Function `screen.clear()`

- Změní barvu všech pixelů na bílou

```
ev3 = EV3Brick()
ev3.screen.draw_text(0, 50,
"ahoj", Color.WHITE,
Color.BLACK)
wait(2000)
ev3.screen.clear()
wait(3000)
```

Function `screen.load_image(source)`

- Na displej je vykreslen obrázek
- `Source` – (`Image` nebo `str`) určuje obrázek, který bude zobrazen na displej. Obrázek může být součástí knihovny nebo proměnná určuje název přiloženého souboru.

```
ev3 = EV3Brick()  
#načtení obrázku z knihovny  
ev3.screen.load_image(ImageFile.BACKWARD)  
wait(3000)  
#načtení obrázku ze souboru  
ev3.screen.load_image("up.png")  
wait(3000)
```

Function `screen.draw_image(x, y, source, transparent=None)`

- Vykreslí obrázek na displej na dané souřadnice
- `X-` (`int`) určuje horizontální souřadnici
- `y-` (`int`) určuje vertikální souřadnici
- `Source` – (`Image` nebo `str`) – určuje obrázek, který bude vykreslen
- `Transparent` – (`Color`) určuje barvu, která bude brána za průhlednou

```
ev3 = EV3Brick()
ev3.screen.draw_image(50,
50, ImageFile.BACKWARD)
wait(3000)

ev3.screen.draw_image(0,
50, "up.png", Color.BLUE)
wait(3000)
```

Function `screen.draw_pixel(x, y, color=Color.BLACK)`

- Obarví daný pixel danou barvou
- `X-` (int) určuje horizontální souřadnici pixelu
- `y-` (int) určuje vertikální souřadnici pixelu
- `Color` – (Color) určuje barvu pixelu

```
ev3 = EV3Brick()
ev3.screen.draw_pixel(50,
50, Color.BLUE)
ev3.screen.draw_pixel(50,
51, Color.BLUE)
ev3.screen.draw_pixel(51,
50, Color.BLUE)
ev3.screen.draw_pixel(51,
51, Color.BLUE)
wait(3000)
```

Function `screen.draw_line(x1, y1, x2, y2, width=1, color=Color.BLACK)`

- Nakreslí úsečku mezi dvěma pixely
- X1 – (int) určuje horizontální souřadnici prvního pixelu
- Y1 – (int) určuje vertikální souřadnici prvního pixelu
- X2 – (int) určuje horizontální souřadnici druhého pixelu
- Y2 – (int) určuje vertikální souřadnici druhého pixelu
- Width – (int) určuje tloušťku čáry
- Color – (Color) určuje barvu čáry

```
ev3 = EV3Brick()
```

```
ev3.screen.draw_line(50, 50, 150,  
100, 3, Color.BLUE)
```

```
wait(3000)
```


Function `screen.draw_box(x1, y1, x2, y2, r=0, fill=False, color=Color.BLACK)`

- Nakreslí obdélník mezi dvěma pixely
- X1 – (int) určuje horizontální souřadnici prvního pixelu
- Y1 – (int) určuje vertikální souřadnici prvního pixelu
- X2 – (int) určuje horizontální souřadnici druhého pixelu
- Y2 – (int) určuje vertikální souřadnici druhého pixelu
- R – (int) určuje zaoblení rohů
- Fill – (bool) určuje, zda bude obdélník vyplněný
- Color – (Color) určuje barvu obdélníku

```
ev3 = EV3Brick()
```

```
ev3.screen.draw_box(50, 50, 150,  
100, 10, True, Color.BLACK)
```

```
wait(3000)
```

Function `screen.draw_circle(x, y, r, fill=False, color=Color.BLACK)`

- Nakreslí kružnici s daným středem a poloměrem
- `X1` – (int) určuje horizontální souřadnici středu
- `Y1` – (int) určuje vertikální souřadnici středu
- `R` – (int) určuje poloměr
- `Fill` – (bool) určuje, zda bude kružnice vyplněná
- `Color` – (Color) určuje barvu kružnice

```
ev3 = EV3Brick()
```

```
ev3.screen.draw_circle(70, 50,  
15, False, Color.BLACK)
```

```
wait(3000)
```

Parametr `screen.width`

- Vrací šířku obrazovky v pixelech (int)

```
ev3 = EV3Brick()
```

```
x=ev3.screen.width/2
```

```
ev3.screen.draw_circle(x, 50,  
15, False, Color.BLACK)
```

```
wait(3000)
```

Parametr screen.height

- Vrací výšku obrazovky v pixelech (int)

```
ev3 = EV3Brick()  
x=ev3.screen.width/2  
y=ev3.screen.height/2  
ev3.screen.draw_circle(x, y,  
15, False, Color.BLACK)  
wait(3000)
```

Function `screen.save(filename)`

- Uloží screenshot obrazovky EV3 jako PNG soubor
- `filename` – (str) určuje název souboru

```
ev3 = EV3Brick()  
x=ev3.screen.width/2  
y=ev3.screen.height/2  
ev3.screen.draw_circle(x, y, 15,  
False, Color.BLACK)  
wait(3000)  
ev3.screen.save('obr')
```



Obrazové stopy

Class Image(source, sub=False)

- Source –(Image nebo str) určuje obrázek, se kterým budeme pracovat
- Sub – (bool) určuje, zda se jedná o celý obrázek nebo jen o jeho část. Je možné přidat nepovinné parametry {x1, y1, x2, y2}
- Vytvoření prázdného obrázku (`empty()`)
- Psaní textu (`draw_text()`, `print()`)
- Změna stylu písma (`set_font()`)
- Vymazání celé obrazovky (`clear()`)
- Vložení obrázku ze souboru (`load_image()`, `draw_image()`)
- Zakreslení pixelu (`draw_pixel()`)
- Zakreslení geometrických útvarů (`draw_line()`, `draw_box()`, `draw_circle()`)
- Zjištění rozměrů obrázku (`height`, `width`)
- Uložení obrázku (`save()`)

Function `draw_text(x, y, text, text_color=Color.BLACK, background_color=None)`

- Zakreslí text na danou pozici
- `X` – (int) určuje horizontální souřadnici umístění textu
- `Y` – (int) určuje vertikální souřadnici umístění textu
- `Text` – (str) určuje text, který má být zakreslen
- `Text_color` – (Color) určuje barvu textu
- `Background_color` – (Color) určuje barvu pozadí

```
ev3 = EV3Brick()  
obr=Image(ImageFile.UP, sub=False)  
obr.draw_text(0, 50,  
"ahoj", Color.WHITE, Color.BLACK)  
wait(2000)
```


Function `print(*args, sep=' ', end='\n')`

- Vypíše text na obrázek s bílým pozadím
- `*args` – (object) určuje objekty, jejichž hodnoty se mají vypsat
- `Sep` – (str) určuje znaky, které mají rozdělovat 2 objekty
- `End` – (str) určuje znaky, které budou vypsány za posledním objektem

```
obr=Image(ImageFile.UP, sub=False)
obr.screen.print("ahoj")
wait(2000)
```

Function set_font(font)

- Změní styl písma
- Font – (Font) určuje proměnnou obsahující vlastnosti písma

```
ev3 = EV3Brick()
obr=Image(ImageFile.UP, sub=False)
#nastaveni fontu
tiny_font = Font(size=6)
big_font = Font(size=24, bold=True)
obr.print('Hello! ')
Obr.set_font(tiny_font)
obr.print('hello ')
obr.set_font(big_font)
obr.print('HELLO')
wait(5000)
```

Function clear()

- Změní barvu všech pixelů na bílou

```
ev3 = EV3Brick()  
obr=Image(ImageFile.UP, sub=False)  
obr.draw_text(0, 50,  
"ahoj", Color.WHITE, Color.BLACK)  
obr.clear()  
wait(3000)
```

Function load_image(source)

- Do obrázku je vložen jiný obrázek
- Source – (Image nebo str) určuje obrázek. Obrázek může být součástí knihovny nebo proměnná určuje název přiloženého souboru.

```
ev3 = EV3Brick()
obr=Image(ImageFile.UP, sub=False)
obr.load_image(ImageFile.BACKWARD)
#načtení obrázku z knihovny
wait(3000)

obr.load_image("up.png") #načtení
obrázku ze souboru
wait(3000)
```

Function `draw_image(x, y, source, transparent=None)`

- Vloží obrázek na dané souřadnice obrázku
- `X- (int)` určuje horizontální souřadnici
- `y- (int)` určuje vertikální souřadnici
- `Source – (Image nebo str)` – určuje obrázek, který bude vykreslen
- `Transparent – (Color)` určuje barvu, která bude brána za průhlednou

```
ev3 = EV3Brick()
obr=Image(ImageFile.UP, sub=False)
obr.draw_image(50,
50, ImageFile.BACKWARD)
wait(3000)
obr.draw_image(0,
50, "up.png", Color.BLUE)
wait(3000)
```

Function `draw_pixel(x, y, color=Color.BLACK)`

- Obarví daný pixel danou barvou
- `x`- (int) určuje horizontální souřadnici pixelu
- `y`- (int) určuje vertikální souřadnici pixelu
- `Color` – (Color) určuje barvu pixelu

```
ev3 = EV3Brick()
obr=Image(ImageFile.UP, sub=False)
obr.draw_pixel(50, 50, Color.BLUE)
obr.draw_pixel(50, 51, Color.BLUE)
obr.draw_pixel(51, 50, Color.BLUE)
obr.draw_pixel(51, 51, Color.BLUE)
wait(3000)
```

Function `draw_line(x1, y1, x2, y2, width=1, color=Color.BLACK)`

- Vytvoří úsečku mezi dvěma pixely
- `X1` – (int) určuje horizontální souřadnici prvního pixelu
- `Y1` – (int) určuje vertikální souřadnici prvního pixelu
- `X2` – (int) určuje horizontální souřadnici druhého pixelu
- `Y2` – (int) určuje vertikální souřadnici druhého pixelu
- `Width` – (int) určuje tloušťku čáry
- `Color` – (Color) určuje barvu čáry

```
ev3 = EV3Brick()  
obr=Image(ImageFile.UP, sub=False)  
obr.draw_line(50, 50, 150, 100,  
3, Color.BLUE)  
wait(3000)
```

Function `draw_box(x1, y1, x2, y2, r=0, fill=False, color=Color.BLACK)`

- Vytvoří obdélník mezi dvěma pixely
- `X1` – (int) určuje horizontální souřadnici prvního pixelu
- `Y1` – (int) určuje vertikální souřadnici prvního pixelu
- `X2` – (int) určuje horizontální souřadnici druhého pixelu
- `Y2` – (int) určuje vertikální souřadnici druhého pixelu
- `R` – (int) určuje zaoblení vrcholů
- `Fill` – (bool) určuje, zda bude obdélník vyplněný
- `Color` – (Color) určuje barvu obdélníku

```
ev3 = EV3Brick()
```

```
obr=Image(ImageFile.UP, sub=False)
```

```
obr.draw_box(50, 50, 150, 100, 10,  
True, Color.BLACK)
```

```
wait(3000)
```


Function `draw_circle(x, y, r, fill=False, color=Color.BLACK)`

- Nakreslí kružnici se daným středem a poloměrem
- `X1` – (int) určuje horizontální souřadnici středu
- `Y1` – (int) určuje vertikální souřadnici středu
- `R` – (int) určuje poloměr
- `Fill` – (bool) určuje, zda bude kružnice vyplněná
- `Color` – (Color) určuje barvu kružnice

```
ev3 = EV3Brick()  
obr=Image(ImageFile.UP, sub=False)  
obr.draw_circle(70, 50, 15, False,  
Color.BLACK)  
wait(3000)
```

Parametr width

- Vrací šířku obrázku v pixelech (int)

```
ev3 = EV3Brick()
obr=Image(ImageFile.UP,sub=False)
x=obr.width/2
obr.draw_circle(x, 50, 15, False,
Color.BLACK)
wait(3000)
```

Parametr height

- Vrací výšku obrázku v pixelech (int)

```
obr=Image(ImageFile.UP, sub=False)
```

```
x=obr.width/2
```

```
y=obr.height/2
```

```
obr.draw_circle(x, y, 15, False,  
Color.BLACK)
```

```
wait(3000)
```

Function save(filename)

- Uloží obrázek jako PNG soubor
- Filename – (str) určuje název souboru

```
obr=Image(ImageFile.UP, sub=False)
```

```
x=obr.width/2
```

```
y=obr.height/2
```

```
obr.draw_circle(x, y, 15, False,  
Color.BLACK)
```

```
wait(3000)
```

```
obr.save('obr')
```



Baterie

EV3 -baterie

- Zjistí napětí baterie ([battery.voltage\(\)](#))
- Zjistí proud baterie([battery.current\(\)](#))

Function `battery.voltage()`

- Vrací hodnotu (int) napětí baterie v milivoltech

```
ev3 = EV3Brick()  
print(ev3.battery.voltage())
```

Function `battery.current()`

- Vrací hodnotu (int) proudu baterie v miliampérech

```
ev3 = EV3Brick()  
print(ev3.battery.current())
```




Motor

Class Motor(port, positive_direction=Direction.CLOCKWISE, gears=None)

- Port – (Port) určuje port, na kterém je motor připojen
- Positive_direction – (Direction) určuje kladný směr otáčení
- Gears – (array of int) – určuje počet ozubených kol při použití převodů
- Roztočení motoru ([run\(\)](#), [run_time\(\)](#), [run_angle\(\)](#), [run_target\(\)](#), [run_until_stalled\(\)](#), [dc\(\)](#))
- Zastavení motoru ([stop\(\)](#), [brake\(\)](#), [hold\(\)](#))
- Informacemi o motoru ([speed\(\)](#), [angle\(\)](#), [reset_angle\(\)](#))

Function run(speed)

- Roztočí servomotor danou rychlostí
- Speed – (int) určuje rychlost, jakou se motor bude otáčet [°/s]

```
mot=Motor(Port.B,  
Direction.COUNTERCLOCKWISE)
```

```
mot.run(100)
```

```
wait(2000)
```

Function `run_time(speed, time, then=Stop.HOLD, wait=True)`

- Roztočí motor konstantní rychlostí po danou dobu
- `speed` – (int) určuje rychlost otáčení motoru [°/s]
- `time` – (int) určuje délku trvání manévru [ms]
- `then` – (Stop) určuje, jestli po zastavení motorů budou motory působit pasivně nebo aktivně proti změně úhlu nebo nebudou působit proti pohybu žádnou silou.
- `Wait` – (bool) určuje, zda program čeká na dokončení funkce `run_time()`

```
mot=Motor(Port.B,  
Direction.COUNTERCLOCKWISE)  
  
mot.run_time(200, 2000, Stop.HOLD,  
True)  
  
wait(3000)
```

Function `run_angle(speed, rotation_angle, then=Stop.HOLD, wait=True)`

- Roztočí motor konstantní rychlostí o daný úhel
- `speed` – (int) určuje rychlost otáčení motoru [$^{\circ}/s$]
- `Angle` – (int) úhel, o který se motor otočí [$^{\circ}$]
- `then` – (Stop) určuje, jestli po zastavení motorů budou motory působit pasivně nebo aktivně proti změně úhlu nebo nebudou působit proti pohybu žádnou silou.
- `Wait` – (bool) určuje, zda program čeká na dokončení funkce `run_angle()`

```
mot=Motor(Port.B,  
Direction.COUNTERCLOCKWISE)
```

```
mot.run_angle(100, 360, then =  
Stop.HOLD, wait= True)
```

```
wait(1000)
```

Function `run_target(speed, target_angle, then=Stop.HOLD, wait=True)`

- Roztočí motor konstantní rychlostí, aby dosáhl požadovaného úhlu
- `speed` – (int) určuje rychlost otáčení motoru [°/s]
- `target__angle` – (int) úhel, kterého chceme dosáhnout [°]
- `then` – (Stop) určuje, jestli po zastavení motorů budou motory působit pasivně nebo aktivně proti změně úhlu nebo nebudou působit proti pohybu žádnou silou.
- `Wait` – (bool) určuje, zda program čeká na dokončení funkce `run_target()`

```
mot=Motor(Port.B, Direction.COUNTERCLOCKWISE)
```

```
#posun o daný úhel, takže se otočí o 360  
stupňů
```

```
mot.run_angle(100, 360, then = Stop.HOLD,  
wait= True)
```

```
wait(1000)
```

```
#posun NA UR CITY uhel, takže pokud z  
minuleho prikazu jsme na uhlu 360 a my  
chceme na 370, #proto se otoci pouze o 10  
stupnu
```

```
mot.run_target(100, 370, then=Stop.HOLD,  
wait=True)
```

Function `run_until_stalled(speed, then=Stop.COAST, duty_limit=None)`

- Otáčí motorem konstantní rychlostí, dokud motor není zastaven vnější silou
- Funkce vrátí úhel zastavení motoru (int)
- `speed` – (int) určuje rychlost otáčení motoru [°/s]
- `then` – (Stop) určuje, jestli po zastavení motorů budou motory působit pasivně nebo aktivně proti změně úhlu nebo nebudou působit proti pohybu žádnou silou.
- `Duty_limit` – (int) určuje, jak velká síla bude považována dostatečnou pro zastavení [%]

```
mot=Motor(Port.B,  
Direction.COUNTERCLOCKWISE)
```

```
uhel=mot.run_until_stalled(200,  
then=Stop.COAST, duty_limit=100)
```

```
#vraci konecny uhel otaceni
```

```
print(uhel)
```

Function dc(duty)

- Roztočí motor jako by byl stejnosměrný
- duty – (int) určuje výkon motoru. [%] Záporné hodnoty využívají stejného výkonu jako jejich kladné ekvivalenty, avšak rotace probíhá na opačnou stranu.

50% výkon na opačnou stranu,
než je motor definovaný

```
mot.dc(-50)
```

```
wait(1000)
```


Function stop()

- Funkce zastaví motor, ale motor dále nepůsobí silou proti změně úhlu. Zastavení je postupné, protože brždění je prováděno pouze pomocí tření.

```
mot. dc(100)
```

```
wait(2000)
```

```
mot. stop()
```

```
wait(2000)
```

Function brake()

- Funkce zastaví motor a dále působí pasivní silou proti změně úhlu. Zastavení je téměř okamžité, protože brždění je prováděno pomocí tření a napětí.

```
mot. dc(100)
```

```
wait(2000)
```

```
mot. brake()
```

```
wait(2000)
```

Function hold()

- Funkce zastaví motor a dále udržuje úhel zastavení servomotoru. Zastavení je téměř okamžité, protože brždění je prováděno pomocí tření a napětí.

```
mot. dc(100)
```

```
wait(2000)
```

```
mot. hold()
```

```
wait(2000)
```

Function speed()

- Vrací rychlost (int) motoru [°/s]

```
mot.dc(100)
```

```
wait(2000)
```

```
print(mot.speed())
```

```
wait(2000)
```

Function angle()

- Vrací úhel (int) motoru [°]

```
mot.dc(100)
```

```
wait(2000)
```

```
print(mot.angle())
```

```
wait(2000)
```

Function `reset_angle(angle)`

- Změní hodnotu úhlu
- `Angle` – (int) určuje hodnotu úhlu po provedení funkce [°]

```
mot.dc(100)
```

```
wait(2000)
```

```
mot.reset_angle(20)
```

```
#vypíše hodnotu úhlu, v tomto  
případě kolem 20
```

```
print(mot.angle())
```

```
wait(2000)
```

Senzory



Dotykový
senzor

Světelný
senzor

Gyroskopický
senzor

Ultrazvukový
senzor



Dotykový senzor

Class TouchSensor(port)

- Port – (Port) určuje port, na kterém je senzor připojen
- Zjistí, zda je senzor sepnutý ([pressed\(\)](#))

Function pressed()

- Vrací hodnotu (bool), zda je senzor sepnutý

```
# namapovani touch senzoru na port 3
dot = TouchSensor(Port.S3)

# true nebo false, jestli je senzor
zmacknuty

print(dot.pressed())
```



Světelný senzor

Class ColorSensor(port)

- Port – (Port) určuje port, na kterém je senzor připojen
- Měření barvy povrchu ([color\(\)](#))
- Měření okolního osvětlení ([ambient\(\)](#))
- Měření odražených paprsků červeného světla od povrchu ([reflection\(\)](#))
- Měření odražených paprsků červeného, zelená a modrého světla od povrchu ([rgb\(\)](#))

Function color()

- Vrací hodnotu (Color nebo None) v podobě barvy povrchu
- Vrací hodnotu None, pokud barva nebyla rozeznána

```
c=ColorSensor(Port.S2)  
print(c.color()) # vraci barvu
```

Function ambient()

- Vrací hodnotu (int) dopadajícího světla na senzor [%]

```
c=ColorSensor(Port.S2)
```

```
# vraci intenzitu osvetleni
```

```
print(c.ambient())
```

Function reflection()

- Vrací hodnotu (int)
odražených paprsků
červeného světla [%]

```
c=ColorSensor(Port.S2)
```

```
# vraci odrazivost cerveneho  
svetla
```

```
print(c.reflection())
```

Function rgb()

- Vrací hodnotu (array of int) odražených paprsků červeného, zeleného a modrého světla [%]

```
c=ColorSensor(Port.S2)
```

```
# vraci odrazivost barev jako  
trojici cisel (R,G,B)
```

```
print(c.rgb())
```




Ultrazvukový senzor

Class UltrasonicSensor(port)

- Port – (Port) určuje port, na kterém je senzor připojen
- Určení vzdálenosti ([distance\(\)](#))
- Zjištění přítomnosti jiných ultrazvukových senzorů ([presence\(\)](#))

Function distance(silent=False)

- Vrací vzdálenost (int) senzoru od překážky [mm]
- Silent – (bool) určuje, zda se senzor po změření vzdálenosti přepne do tichého režimu a nebude vysílat ultrazvukové vlny.

```
u=UltrasonicSensor(Port.S1)  
print(u.distance(silent=False))
```

Function presence()

- Vrací hodnotu (bool), zda byly detekovány ultrazvukové vlny jiného ultrazvukového senzoru

```
u=UltrasonicSensor(Port.S1)  
print(u.presence())
```

The background is a dark grey-blue color. In the four corners, there are decorative white line-art patterns resembling circuit board traces or neural network connections. These patterns consist of thin lines that branch out and terminate in small circles, creating a grid-like but irregular structure.

Gyroskopický senzor

Class GyroSensor(port, positive_direction=Direction.CLOCKWISE)

- Port – (Port) určuje port, na kterém je senzor připojen
- Positive_direction – (Direction) určuje kladný směr rotace
- Určení úhlu ([angle\(\)](#))
- Určení úhlové rychlosti ([speed\(\)](#))
- Změna hodnoty úhlu ([reset_angle\(\)](#))

Function angle()

- Vrací úhel (int)
gyroskopického senzoru [°]

```
g=GyroSensor(Port.S3)
#urcuje uhel senzoru
print(g.angle())
```

Function speed()

- Vrací úhlovou rychlost (int) senzoru [$^{\circ}/s$]
- Při zavolání funkce speed() se vynuluje úhel senzoru, proto není vhodné využívat tyto funkce ve stejném kódu

```
g=GyroSensor(Port.S3)
```

```
#urcuje rychlost rotace senzoru  
podle sipek na nem
```

```
print(g.speed())
```


Function reset_angle()

- Mění hodnotu úhlu senzoru
- Angle – (int) určuje hodnotu nastavovaného úhlu [°]

```
g=GyroSensor(Port.S3)
#zresetuje uhel na 100
g.reset_angle(100)
print(g.angle())
```



**Komplexní
funkce pro
pohyb
robota**

Class DriveBase (left_motor, right_motor, wheel_diameter, axle_track)

- Left_motor – (Class) určuje třídu pro levý motor robota
- Right_motor – (Class) určuje třídu pro pravý motor robota
- Wheel_diameter – (int) určuje průměr kol [mm]
- Axle_track – (int) určuje vzdálenost mezi body dotyku kol se zemí [mm]
- Rozpohybuje robota ([straight\(\)](#), [turn\(\)](#), [drive\(\)](#))
- Podává informace o robotovi ([distance\(\)](#), [angle\(\)](#), [state\(\)](#))
- Mění parametry robota ([settings\(\)](#), [reset\(\)](#))
- Zastaví robota ([stop\(\)](#))

Function straight(distance)

- Robot ujede danou vzdálenost bez zatáčení
- Distance – (int) určuje vzdálenost, která má být robotem uražena [mm]

```
motR=Motor(Port.B,  
Direction.CLOCKWISE)
```

```
motL=Motor(Port.C,  
Direction.CLOCKWISE)
```

```
rob=DriveBase(motL, motR, 68.7, 169)
```

```
rob.straight(1000) #ujede 1 metr
```

Function turn(angle)

- Robot se na místě otočí o daný úhel angle.
- Angle – (int) určuje úhel, o který se má robot otočit. [°]

```
motR=Motor(Port.B,  
Direction.CLOCKWISE)
```

```
motL=Motor(Port.C,  
Direction.CLOCKWISE)
```

```
rob=DriveBase(motL, motR, 68.7, 169)
```

```
# otoci se na miste o 360 stupnu
```

```
rob.turn(360)
```

Function `drive(drive_speed, turn_rate)`

- Robot se pohybuje rychlostí `drive_speed` otáčí se rychlostí `turn_rate`.
- `Drive_speed` – (int) určuje rychlost, kterou se robot pohybuje [mm/s]
- `Turn_rate` – (int) určuje rychlost rotace robota [°/s]

```
motR=Motor(Port.B,  
Direction.CLOCKWISE)
```

```
motL=Motor(Port.C,  
Direction.CLOCKWISE)
```

```
rob=DriveBase(motL, motR, 68.7, 169)
```

```
#jede 100 mm/s a otaci se 20 ° /s
```

```
rob.drive(100, 20)
```

Function settings(straight_speed, straight_acceleration, turn_rate, turn_acceleration)

- Nastavení maximální rychlost a zrychlení translace robota. Nastaví maximální rychlost a zrychlení rotace robota
- Straight_speed – (int) určuje maximální rychlost translace [mm/s]
- Straight_acceleration – (int) určuje maximální zrychlení translace [mm/s²]
- Turn_rate – (int) určuje maximální rychlost rotace [°/s]
- Turn_acceleration – (int) určuje maximální zrychlení rotace [°/s²]

```
motR=Motor(Port.B,  
Direction.CLOCKWISE)
```

```
motL=Motor(Port.C,  
Direction.CLOCKWISE)
```

```
rob=DriveBase(motL, motR, 68.7, 169)
```

```
rob.settings(150, 50, 80, 30)
```

```
rob.turn(360) # otoc se na miste o  
360 stupnu
```

Function stop()

- Zastaví robota. Po zastavení se kola mohou volně otáčet.

```
motR=Motor(Port.B,  
Direction.CLOCKWISE)
```

```
motL=Motor(Port.C,  
Direction.CLOCKWISE)
```

```
rob=DriveBase(motL, motR, 68.7, 169)
```

```
rob.drive(70, -10)
```

```
wait(1000)
```

```
rob.stop()
```

```
wait(2000)
```


Function reset()

- Funkce vynuluje ураženou vzdálenost a rotaci robota.

```
motR=Motor(Port.B, Direction.CLOCKWISE)
motL=Motor(Port.C, Direction.CLOCKWISE)
rob=DriveBase(motL, motR, 68.7, 169)
rob.drive(70,-10)
wait(1000)
rob.stop()
wait(2000)
rob.reset()
print(rob.distance())
```

Function distance()

- Vrací vzdálenost (int) uraženou od posledního zavolání funkce reset() nebo od začátku kódu [mm]

```
motR=Motor(Port.B, Direction.CLOCKWISE)
motL=Motor(Port.C, Direction.CLOCKWISE)
rob=DriveBase(motL, motR, 68.7, 169)
rob.drive(70,-10)
wait(1000)
rob.stop()
wait(2000)
print(rob.distance())
rob.reset()
print(rob.distance())
```

Function angle()

- Vrací úhel, o který se robot otočil od posledního zavolání funkce reset() nebo od začátku kódu.

```
motR=Motor(Port.B, Direction.CLOCKWISE)
motL=Motor(Port.C, Direction.CLOCKWISE)
rob=DriveBase(motL, motR, 68.7, 169)
rob.drive(70,-10)
wait(1000)
rob.stop()
wait(2000)
print(rob.angle())
rob.reset()
print(rob.angle())
```

Function state()

- Vrací hodnoty (array of int) v poli o 4 hodnotách ve tvaru [uražená_vzdálenost[mm], rychlost_translace[mm/s], úhel_robota[°], rychlost_rotace[°/s]]

```
motR=Motor(Port.B, Direction.CLOCKWISE)
motL=Motor(Port.C, Direction.CLOCKWISE)
rob=DriveBase(motL, motR, 68.7, 169)
rob.drive(70,-10)
wait(1000)
print(rob.state())
wait(2000)
rob.stop()
wait(100)
print(rob.state())
```



Záznam dat

Class DataLog(*headers, name='log', timestamp=True, extension='csv', append=False)

- Vytvoří nový soubor
- *header – (str) určuje jména sloupců, která se v souboru vytvoří
- Name – (str) určuje název souboru
- Timestamp – (bool) určuje, zda má být za název přidána informace o datu a času
- Extension – (str) určuje příponu souboru
- Append – (bool) určuje, zda má být existující soubor otevřen, nebo nahrazen
- Vytvoří záznam na nový řádek ([log\(\)](#))

Function log(*values)

- Vytvoří záznam v souboru na nový řádek
- *values – (objects) určuje, jaké hodnoty budou zapsány na jeden řádek

```
data = DataLog('cas',  
'uhel', name='rozbehMotoru', timestamp=False,  
extension='csv', append=False)
```

```
mot = Motor(Port.B)
```

```
mot.run(500)
```

```
stopky = Stopwatch()
```

```
for i in range(50):
```

```
    uhel = mot.angle()
```

```
    cas = stopky.time()
```

```
    data.log(cas, uhel)
```

```
    wait(100)
```



Bluetooth komunikace

Před navázáním spojení pomocí kódu

- Zapnout Bluetooth (v menu *Wireless and Networks* → *Bluetooth* → *Powered*)
- Zviditelnit zařízení (v menu *Wireless and Networks* → *Bluetooth* → *Visible*)
- Nalézt a spárovat zařízení přes Bluetooth
- První se musí pustit EV3 chovající se jako server

Class BluetoothMailboxServer()

- Vytvoří objekt reprezentující Bluetooth spojení jako server
- Čekání na spojení ([wait for connection\(\)](#))

Function `wait_for_connection()`

- Vyčkává, dokud se EV3 chovající se jako klient nepřipojí k serveru

```
from pybricks.messaging import  
BluetoothMailboxServer  
  
server =  
BluetoothMailboxServer()  
  
server.wait_for_connection()
```

Class BluetoothMailboxClient()

- Vytvoří objekt reprezentující Bluetooth spojení jako klient
- Navázat spojení ([connect\(\)](#))

Function connect()

- Připojí se ke kostce EV3 chovající se jako server

```
from pybricks.messaging import  
BluetoothMailboxClient  
  
client =  
BluetoothMailboxClient()  
client.connect('ev3dev')
```

Typy schránek

- Pro správnou komunikace je nutné na obou zařízeních pojmenovat schránku stejně
- Class Mailbox(name, connection, encode=None, decode=None)
 - Posílání bytů
- Class LogicMailBox(name, connection)
 - Posílání bool hodnot (True/False)
- Class NumericMailBox(name, connection)
 - Posílání čísel
- Class TextMailBox(name, connection)
 - Posílání textu
- Name –(str) název schránky
- Connection – (object) určuje chování jako server nebo klient

Function read()

- Vrací hodnotu ve schránce nebo hodnotu None, pokud je schránka prázdná

```
from pybricks.messaging import  
BluetoothMailboxClient  
  
client =  
BluetoothMailboxClient()  
  
mbox = TextMailbox('greeting',  
client)  
  
client.connect('ev3dev')  
print(mbox.read())
```

Function `send(value, brick=None)`

- Pošle zprávu pomocí Bluetooth komunikace
- Value – obsah zprávy, která se má poslat
- Brick – (str) označuje EV3, které se má daná zpráva poslat. Pokud je hodnota None, pošle se zpráva všem připojeným EV3.

```
from pybricks.messaging import  
BluetoothMailboxClient  
  
client = BluetoothMailboxClient()  
mbox = TextMailbox('greeting', client)  
client.connect('ev3dev') mbox.send('hello  
to you!')
```


Function wait()

- Vyčkává, dokud nepřijde nová zpráva do schránky

```
from pybricks.messaging import  
BluetoothMailboxClient  
  
client = BluetoothMailboxClient()  
mbox = TextMailbox('greeting',  
client)  
  
client.connect('ev3dev')  
mbox.wait()
```

Function wait_new()

- Vyčkává, dokud nepřijde nová zpráva do schránky. Navíc obsah zprávy musí být rozdílný od současné hodnoty ve schránce

```
from pybricks.messaging import  
BluetoothMailboxClient  
  
client = BluetoothMailboxClient()  
mbox = TextMailbox('greeting',  
client)  
  
client.connect('ev3dev')  
mbox.wait_new()
```