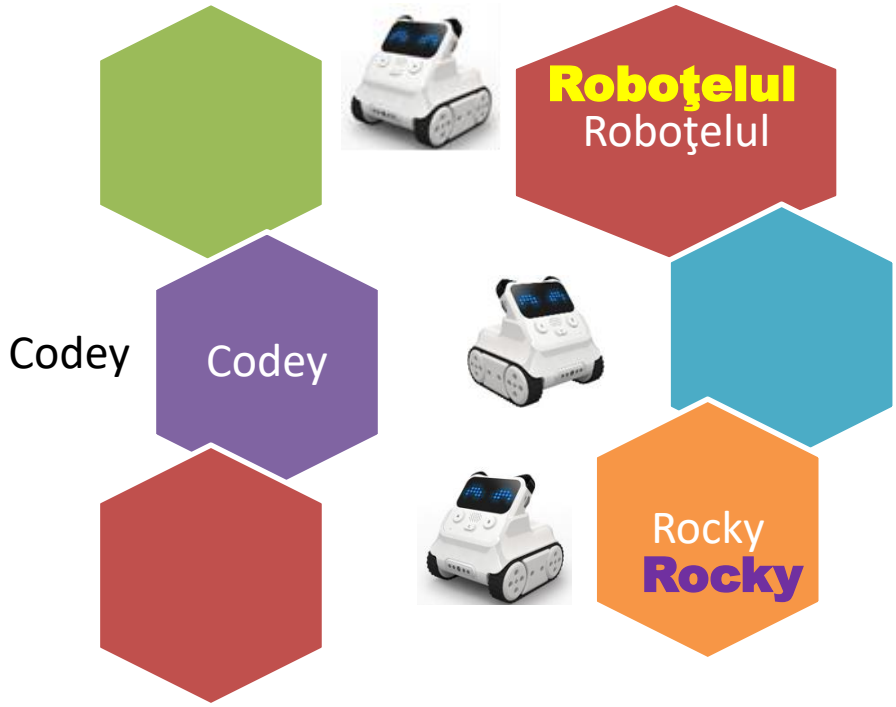
UNIVERSITATEA PEDAGOGICĂ DE STAT „ION CREANGĂ”
DIN CHIŞINĂU

Teodora VASCAN



Chișinău 2023

CZU

Aprobat pentru tipar de Senatul Universității Pedagogice de Stat „Ion Creangă”

Lucrare realizată în cadrul proiectului de cercetări științifice „Metodologia implementării TIC în procesul de studiere a științelor reale în sistemul de educație din Republica Moldova din perspectiva inter/transdisciplinarității (*concept STEAM*)*”, inclus în „Program de stat”* (2020-2023)*, Prioritatea* IV*: Provocări societale, cifrul* 20.80009.0807.20*, cu suportul financiar oferit de Agenția Națională pentru Dezvoltare și Cercetare.*

Recenzenți:

Inga Camerzan, doctor, conferențiar universitar, director Institutul de Matematică și Informatică „Vladimir Andrunachievici”

Veverița Tatiana, doctor, conferențiar universitar, Universitatea Pedagogică de Stat „ Ion Creangă“

DESCRIEREA CIP A CAMEREI NAȚIONALE A CĂRȚII DIN REPUBLICA MOLDOVA

Vascan, Teodora.
Roboțelul Codey Rocky : Ghid de utilizare / Teodora Vascan ; Universitatea Pedagogică de Stat "Ion Creangă" din Chişinău, Laboratorul de Cercetare "Inteligenţa Artificială Creativă". – Chişinău : [S. n.], 2023 (CEP UPSC) – . – ISBN 978-9975-76-408-7.
Partea 2 : Programarea în Python. – 2023. – 90 p. : fig., tab. color. – Bibliogr.: p. 90 (11 tit.). – Apare cu suportul financiar oferit de Agenția Națională pentru Dezvoltare și Cercetare. – [100] ex. – ISBN 978-9975-46-786-5.
37.016:004.896(075)
V-31

Centrul Editorial-Poligrafic al Universității Pedagogice de Stat „Ion Creangă” din Chișinău, str. Ion Creangă, nr. 1, MD-2069

Cuprins:

Cuvânt înainte

Roboții au fost recunoscuți ca având potențialul de a transforma și îmbunătăți procesul de învățare în educație. Prin experimente practice, astfel de tehnologii pot ajuta elevii să traducă concepte abstracte de matematică și știință în aplicații concrete din lumea reală. Acest ghid sprijină utilizarea roboticii educaționale pentru a crește performanța academică în domenii specifice de concept STEM, strâns aliniate cu subiectele educației formale. Robotica încurajează, de asemenea, rezolvarea problemelor și promovează învățarea prin cooperare.

Prezentul ghid a fost elaborat cu intenția de a putea fi folosit ca material didactic pentru predare a lecțiilor de robotică incluse în curriculumul disciplinei opţionale de studiu Robotica în învăţământul primar, gimnazial şi liceal din Republica Moldova. În special, acesta este dedicat profesorilor și elevilor din gimnaziu și liceu care doresc să practice programarea unui robot cu funcționalități deosebite - Codey Rocky. Scopul acestui ghid este de a ajuta profesorii din învățământul gimnazial și liceal din Republica Moldova, care în mare parte, nu sunt familiarizați cu programarea roboților. În acest context, pentru a veni în sprijinul profesorilor care predau Educația digitală la clasele primare și Informatica la ciclul gimnaziu, a fost elaborat ghidul de utilizare a robotelului Codey Rocky (*Partea I: mediul de programare mBlock)* și acum propunem *Partea II: programarea în Python*, un limbaj de programare de nivel înalt frecvent utilizat cu o sintaxă foarte simplă. În acest ghid s-a ținut cont de domeniile practic noi așa ca *Inteligența Artificială* și *Internetul lucrurilor*. Blocurile *Neuron*, permit elevilor să obțină o înțelegere mai intuitivă a modului în care funcționează senzorii. *Neuron* facilitează proiectarea, crearea și afișarea proiectelor. Cu ajutorul acestui kit, profesorii pot dezvolta cursuri de programare, AI și IoT, și pot organiza competiții de robotică creative.

Exemplele incluse în acest ghid vor controbui la dezvoltarea propriilor proiecte fiind ca o sursă de inspirație.

Autorul

Introducere

Python este un limbaj de programare la nivel înalt, interpretat, orientat pe obiecte, cu semantică dinamică. Structurile sale de date de nivel înalt, combinate cu tastarea dinamică și legarea dinamică, îl fac foarte atractiv pentru dezvoltarea rapidă a aplicațiilor, precum și pentru utilizare ca limbaj de scriptare sau lipire pentru a conecta componentele existente între ele. Sintaxa simplă și ușor de învățat a lui *Python* accentuează lizibilitatea și, prin urmare, reduce costurile de întreținere a programului.

Python este folosit peste tot, de la aplicațiile obișnuite care sunt folosite zilnic, până la creierul unui robot, limbajul este aplicabil în orice fel de situații și este limbajul preferat de mulți programatori.

Python este, de asemenea, ușor de învățat, deoarece este un limbaj de programare interpretat. Aceasta înseamnă că putem rula fiecare linie de cod odată ce am terminat-o de scris, permițându-ne să o verificăm imediat și să facem ajustări dacă este necesar - un mare ajutor pentru cei care încă învață și o economisire de timp pentru programatorii de pretutindeni.

Din aceste considerente, ne-am propus să elaborăm acest ghid, în care să sedcriem posibilitățile de programare a robotelului *Codey Rocky* în limbajul de programare *Python*. Pentru a programa în *Python* pe *Codey Rocky* nu este necesar de a utiliza un alt soft, deoarece putem folosi mediul de programare mBlock în care vom selecta programarea în Python (figura 1).

Utilizarea limbajului Python

Codey este un robot bazat pe limbajul *Python*. Pentru a putea utiliza limbajul *Python*, trebuie de schimbat modul de programare de la „*Blocks*” la „*Python*” (fig. 1).

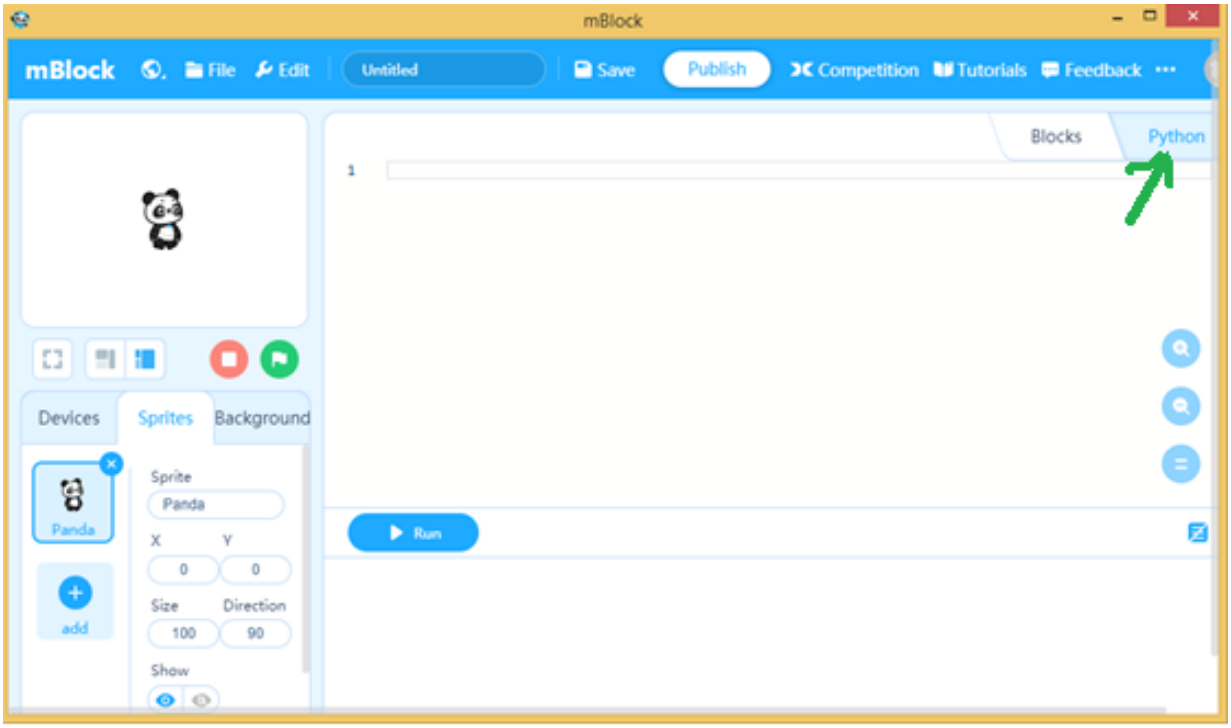


Figura 1: Setarea modului de programare Python

Notă: Ne asigurăm ca Codey să fie selectat fig. 2.

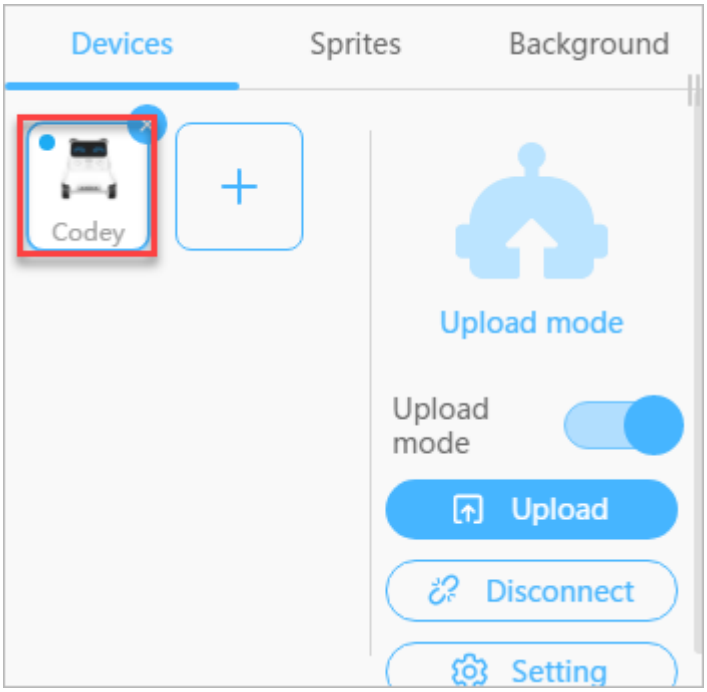


Figura 2: Selectarea și conectarea lui *Codey*

După scrierea codului de program este necesar de a efectua un click pe „*Upload*” pentru a încărca programul în *Codey*.

Convertirea blocurilor în limbaj Python

mBlock permite de a converti programele scrise cu ajutorul blocurilor în limbajul *Python*. Acest lucru este binevenit în cazurile când poate să apară careva probleme la scrierea codului în Python, acesta poate fi scris cu ajutorul blocurilor apoi de convertit în Python. Acest lucru este foarte ușor de realizat și anume:

În zona *Scripturi*, facem clic pe </> pentru a converti script-ul format din blocuri în cod *Python*. Următoarea figură ilustrează un exemplu:

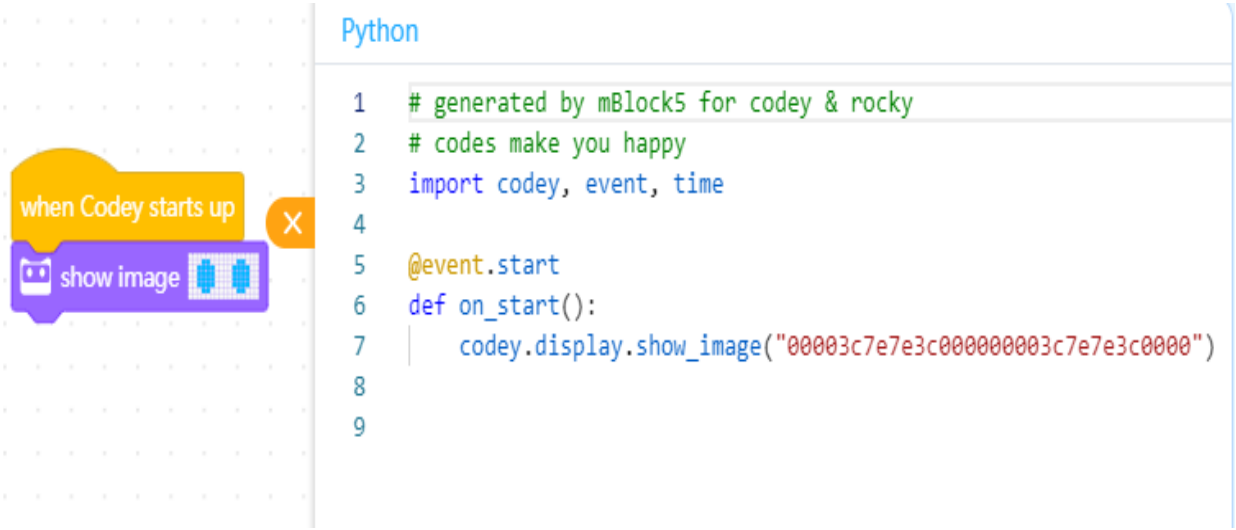


Figura 3: Convertirea blocurilor în cod *Python*

Utilizarea Panoului frontal

În tabelul 1 sunt enumerate funcțiile *Python* cu acțiunile acestora care permit utilizarea Panoului frontal al lui *Codey*.

Tabelul 1: Funcțiile *Python* ce permit utilizarea Panoului frontal al lui *Codey*

<table>
<tr><th>Funcție</th><th>Acțiune</th></tr>
<tr><td>display.show(text, wait=False)</td><td>Afișarea unui text pe Panoul frontal al lui Codey.</td></tr>
<tr><td colspan="2">Exemplu:
<pre>import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.display.show("Salutare", wait=False)</pre></td></tr>
<tr><td>display.show_image(image, time)</td><td>Afișarea unei imagini pe panoul frontal al lui Codey. Imaginea afișată trebuie convertită în date hexazecimale.
Dacă pentru atributul time este indicat orice alt număr diferit de zero, panoul se va opri după durata setată.</td></tr>
<tr><td colspan="2">Exemplu:
<pre>import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.display.show_image("00003c7e7e3c000000003c7e7e3c0000", time_s=15)</pre></td></tr>
</table>

display.show_image(image, x, y)	Afișarea unei imagini pe panoul frontal al lui *Codey* la o anumită poziție indicată de coordonatele (x, y).

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.display.show_image("000c18181c0c000000000c1c18180c00"
, 0, 0)
```

codey.display.set_pixel(x, y, status) / codey.display.toggle_pixel(x, y)	• *set_pixel*: setează starea pixelului la (*x*, *y*); setarea *True* pentru a aprinde pixelul sau *False* pentru a-l opri; • *toggle_pixel*: comută un pixel la (*x*, *y*): *Turn On* dacă este dezactivat și *Off* dacă este activat

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.display.set_pixel(5, 5, True) # aprinde pixelul în
poziția  (5, 5)
codey.display.set_pixel(5, 5, False) # stinge pixelul din
poziția (5, 5)
codey.display.toggle_pixel(7, 7) # comută pixelul în
poziția (7, 7)
```

display.get_pixel(x, y)	Verifică dacă un anumit pixel al lui *Codey* este activat sau nu. Dacă este

	activat, întoarce *True*; altfel returnează *False*

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass

print(codey.display.get_pixel(7, 7))  # printează True dacă pixelul din poziția (7, 7) este aprins
```

display.clear()	Șterge tot conținutul de pe panoul frontal al lui *Codey*.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.display.clear() # Curăță panoul Codey
```

Utilizarea LED-ului RGB de pe Codey

În tabelul 2 sunt enumerate funcțiile *Python* cu acțiunile acestora care permit utilizarea LED-ului RGB al lui *Codey*.

Tabelul 2: Funcțiile *Python* ce permit utilizarea LED-ului RGB al lui *Codey*

<table>
<tr><th>Funcție</th><th>Acțiune</th></tr>
<tr><td>led.show(r, g, b, time)</td><td>Setarea culorilor LED-ului Codey</td></tr>
<tr><td colspan="2">Exemplu: Dacă durata este un număr diferit de zero, LED-ul se va stinge după un anumit număr de secunde specificat.
<pre>import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.led.show(255, 255, 0, 1) # setarea culorii verzi, închidere după o secundă
codey.led.show(255, 0, 0) # schimbă culoarea în roșu</pre></td></tr>
<tr><td>led_off()</td><td>Oprirea LED-ului Codey</td></tr>
<tr><td colspan="2">Exemplu:
<pre>import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.led_off() # stingerea luminii LED-ului</pre></td></tr>
<tr><td>led.set_red(value) /
led.set_green(value) /
led.set_blue(value)</td><td>Setarea valoarii RGB a LED-ului independent. Intervalul fiecărei valori de culoare este 0-255.</td></tr>
<tr><td colspan="2">Exemplu:
<pre>import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.led.set_red(255)
codey.led.set_green(255)
codey.led.set_blue(255)</pre></td></tr>
</table>

Generarea sunetelor

Codey Rocky poate fi programat pentru a emite careva sunete. Evident nu poate emite orișice suntete, deaceia mai jos sunt listate fișierele sonore care pot fi folosite în acest scop.

Lista de fișiere sonore valabile:

- hello.wav - hello
- hi.wav - hi
- bye.wav - bye
- yeah.wav - yeah
- wow.wav - wow
- laugh.wav - laugh
- hum.wav - hum
- sad.wav - sad
- sigh.wav - sigh
- annoyed.wav - annoyed
- angry.wav - angry
- surprised.wav - scared
- yummy.wav - pettish
- curious.wav - curious
- embarrassed.wav - embarrassed
- ready.wav - ready
- sprint.wav - sprint
- sleepy.wav - snore
- meow.wav - meow
- start.wav - start
- switch.wav - switch
- beeps.wav - beeps
- buzzing.wav - buzz
- exhaust.wav - air-out
- explosion.wav - explosion
- gotcha.wav - gotcha
- hurt.wav - painful
- jump.wav - jump
- laser.wav - laser
- level up.wav - level-up
- low energy.wav - low-energy
- metal clash.wav - metal-clash

- prompt tone.wav - prompt-tone
- right.wav - right
- wrong.wav - wrong
- ring.wav - ringtone
- score.wav - score
- shot.wav - shot
- step_1.wav - step_1
- step_2.wav - step_2
- wake.wav - activate
- warning.wav - warning

În tabelul 3 sunt enumerate funcțiile *Python* cu acțiunile acestora care permit generarea sunetelor de către *Codey*.

Tabelul 3: Funcțiile *Pytnon* ce permit generarea sunetelor de către *Codey*

<table>
<tr><th>Funcția</th><th>Acțiune</th></tr>
<tr><td>speaker.play_melody(name)</td><td>Redarea fișierelor audio.</td></tr>
<tr><td colspan="2">Exemplu:
<pre>import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.speaker.play_melody("meow")
# sau
codey.speaker.play_melody("meow.wav") # .wav este opțional</pre></td></tr>
<tr><td>speaker.play_note(note_number, beats)</td><td>Generarea unei note muzicale pentru o durată de timp.
Numărul notei va fi numărul MIDI. De asemenea, putem utiliza nume de note precum „C3” sau „D4”. Numele incorect al notei va produce o eroare în consolă.</td></tr>
<tr><td colspan="2">Exemplu:
<pre>import codey, rocky, time, event
@event.start
def on_start():
    pass</pre></td></tr>
</table>

```
codey.speaker.play_note(36, 1)
codey.speaker.play_note('E5', 0.25)
```

speaker.play_tone(frequency, beats)	Redă o anumită frecvență a sunetului pe o durată de anumite bătăi.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.speaker.play_tone(500, 1)
```

speaker.rest(beats)	Întrerupe sunetul pentru un anumit număr de bătăi.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.speaker.play_tone(500, 1)
codey.speaker.rest(0.25)
```

speaker.stop_sounds()	Oprește toate sunetele.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.speaker.play_tone(500, 1)
codey.speaker.rest(0.25)
codey.speaker.stop_sounds()
```

speaker.volume	Setarea volumului difuzorului *Codey*.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.speaker.volume = (40) # Setarea volumului lui Codey
la 40%
print(codey.speaker.volume) # Redă volumul curent al lui
Codey
codey.speaker.play_tone(500, 1)
```

Utilizarea senzorilor lui Codey

În tabelul 4 sunt enumerate funcțiile *Python* cu acțiunile acestora care permit utilizarea senzorilor lui *Codey*.

Tabelul 4: Funcțiile *Python* ce permit utilizarea senzorilor lui *Codey*

<table>
<tr><th>Funcție</th><th>Acțiune</th></tr>
<tr><td>button_a.is_pressed()/
button_b.is_pressed()/
button_c.is_pressed()</td><td>Dacă butonul specificat (a, b sau c) de pe Codey este apăsat, returnează True; în caz contrar, întoarce False.</td></tr>
<tr><td colspan="2">Exemplu:
<pre>import codey, rocky, time, event
@event.start
def on_start():
    pass
 print(codey.button_a.is_pressed()) # printează True dacă butonul A este tastat</pre></td></tr>
<tr><td>potentiometer.get_value()</td><td>Întoarce poziția butonului potențiometrului lui Codey. Valoarea poate fi de la 0-100.</td></tr>
<tr><td colspan="2">Exemplu:
<pre>import codey, rocky, time, event
@event.start
def on_start():
    pass

print(codey.display.show(potentiometer.get_value()) # Printează poziția potențiometrului</pre></td></tr>
<tr><td>sound_sensor.get_loudness()</td><td>Întoarce intensitatea senzorului de sunet. Valorile vor fi cuprinse între 0-100.</td></tr>
<tr><td colspan="2">Exemplu:
<pre>import codey, rocky, time, event
@event.start
def on_start():</pre></td></tr>
</table>

```
    pass
  print(codey.sound_sensor.get_loudness()) # Întoarce
intensitatea senzorului de sunet
```

light_sensor.get_value()	Întoarce intensitatea luminii detectată de senzorul de lumină. Valorile admisibile: 0-100.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
  print(codey.light_sensor.get_value())
```

motion_sensor.is_shaked()	Spune dacă *Codey* este zdruncinat. Rezultatul este *True* sau *False*.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
    print(codey.motion_sensor.is_shaked()) #Dacă Codey este
zguduit întoarce valoarea True
```

motion_sensor.is_tilted_left()/ motion_sensor.is_tilted_right()/ motion_sensor.is_ears_up()/ motion_sensor.is_ears_down()	Spune dacă *Codey* este înclinat spre dreapta / stânga sau dacă este așezat cu urechile în sus / cu urechile în jos. Rezultatul este *True* sau *False*.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
print(codey.motion_sensor.is_tilted_left()) # Dacă Codey
este înclinat spre stânga, întoarce True
print(codey.motion_sensor.is_ears_up()) # Dacă Codey este c
urechile în sus, întoarce True
```

motion_sensor.get_roll() **motion_sensor.get_pitch()** **motion_sensor.get_yaw()**	Obținem valoarea rulării, înălțimii sau girației giroscopului *Codey*

motion_sensor.get_rotation(axis): axa x, y sau z **motion_sensor.reset_rotation(axis="all") : resetarea unghiurilor de rotație ale giroscopului**	Obținem valoarea de rotație a giroscopului (în grade) în jurul unei anumite axe.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
print("Giratia și pitch-ul isunt:",
codey.motion_sensor.get_yaw(),
codey.motion_sensor.get_pitch()) # întoarce valoarea
giratiei și a pitch-ului a giroscopului
print("Valoarea rotației este:",
codey.motion_sensor.get_rotation(x),
codey.motion_sensor.get_rotation(y),
codey.motion_sensor.get_rotation(z))
codey.motion_sensor.reset_rotation() # resetarea valorii
rotației.
print("Se agită forța:",
codey.motion_sensor.get_shake_strength())
```

get_timer()	Obținem valoarea temporizatorului în câteva secunde (de la pornire sau ultima resetare).

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
codey.display.show(codey.get_timer())
```

reset_timer()	Resetarea timer-ului

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start(): pass
codey.reset_timer()
codey.display.show(codey.get_timer()) # va genera valoarea 0
```

Evenimente și control al fluxului Codey

Codey acceptă evenimente cum ar fi atunci când butonul este apăsat și altele.

Dacă dorim să utilizăm un eveniment, este necesar de declarat o funcție și să o programăm la un anumit eveniment. Un program poate înregistra doar cel mult 6 funcții de eveniment.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
def on_button_a_pressed(): # definește o funcție
    print("Butonul A este tastat!")
codey.on_button_a_pressed() # înregistrează evenimentul
"când butonul A este apăsat"
```

În tabelul 5 sunt descrise evenimentele *Codey* cu acțiunile acestora.

Tabelul 5: Evenimentele *Codey*

<table>
<tr><th>Evenimentul</th><th>Acțiune</th></tr>
<tr><td>on_button_a_pressed()/
on_button_b_pressed()/
on_button_c_pressed()</td><td>Când butonul este apăsat, rulează funcția.</td></tr>
<tr><td colspan="2">Exemplu:
<pre>import codey, rocky, time, event
@event.start
def on_start():
    pass
def on_button_b_pressed(): # definește o funcție
    print("Butonul B este tastat!")
codey.on_button_b_pressed() # înregistrează evenimentul
"când butonul B este apăsat"</pre></td></tr>
<tr><td>on_shaked()</td><td>Când Codey este zdruncinat, apelează funcția.</td></tr>
<tr><td colspan="2">Exemplu:
<pre>import codey, rocky, time, event</pre></td></tr>
</table>

```
@event.start
def on_start():
    pass
def on_shaked():
    print("Sunt zdruncinat!")
```

on_tilted_left()	Apelează funcția când *Codey* se înclină spre stânga.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
def on_tilted_left():
    print("Sunt înclinat-stânga!")
codey.on_tilted_left()
```

on_greater_than(volume, 'sound_sensor')	Când sunetul depășește o anumită valoare, apelează funcția.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass

def on_greater_than():
    print("Volumul este peste 50! Prea tare!")
codey.on_greater_than(50, 'sound_sensor')
```

on_less_than(lightness, 'light_sensor')	Când luminozitatea este sub o anumită valoare, apelează funcția.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass

def on_less_than():
    print("Lumina este sub 30! Prea intuneric!")

codey.on_less_than(30, 'light_sensor')
```

on_received(message_name)	Când se recepționează o transmisie *Scratch*, declanșează funcția.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass

def on_received():
    print("Începe jocul!")
codey.on_received(' Începe jocul!')
```

broadcast(message_name)	Transmite un anumit mesaj către sistemul **Scratch**.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass

def on_received():
    print("Începe jocul!")

codey.on_received(' Începe jocul!')
codey.broadcast(' Începe jocul!')
```

stop_all_scripts()/ stop_this_script()/ stop_other_scripts()	Oprește toate procesele, acest proces sau alte procese.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass

def on_received():
    print("Începe jocul!")

codey.on_received(' Începe jocul!')
codey.broadcast(' Începe jocul!')
codey.stop_all_scripts() # oprește toate procesele pe Codey
```

Mișcarea lui Rocky

În cele ce urmează vom descrie funcțiile responsabile de mișcarea lui *Codey Rocky*. Evident pentru ca *Codey* să se poată deplasa, el are nevoie de platforma *Rocky*.

În tabelul 6 sunt descrise funcțiile *Python* ce pot fi utilizate pentru a-l pune în mișcare pe Codey Rocky.

Tabelul 6: Funcțiile *Python* pentru mișcarea lui *Rocky*

<table>
<tr><th>Funcții</th><th>Acțiune</th></tr>
<tr><td>forward(speed, time)</td><td>Mișcarea înainte pentru o anumită perioadă de timp. Dacă timpul nu este setat, va continua să se miște încontinuu. Valorile admisibile pentru viteză: 0-100.</td></tr>
<tr><td colspan="2">Exemplu:
<pre>import codey, rocky, time, event
@event.start
def on_start():
    pass

rocky.forward(50, 1) # Merge înainte timp de o secundă cu viteza 50
rocky.forward(100) # Merge înainte tot timpul cu viteza 100</pre></td></tr>
<tr><td>backward(speed, time)</td><td>Mișcarea înapoi pentru o anumită perioadă de timp. Dacă timpul nu este setat, va continua să se miște. Valorile admisibile pentru viteză: 0-100.</td></tr>
<tr><td colspan="2">Exemplu:
<pre>import codey, rocky, time, event
@event.start
def on_start():
    pass</pre></td></tr>
</table>

```
rocky.backward(50,2) # Merge înapoi cu viteza 50 timp de 2 secunde
rocky.backward(100) # Merge înapoi cu viteza 100
```

turn_right(speed, time)	Virarea la dreapta pentru o anumită perioadă de timp. Dacă timpul nu este setat, va continua să se rotească. Valorile admisibile pentru viteză: 0-100.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass

rocky.turn_right(50, 1) # Virează la dreapta cu viteza 50 timp de o secundă
rocky.turn_right(100) # Virează la dreapta cu viteza maximă 100
```

turn_left(speed, time)	Virarea la stânga pentru o anumită perioadă de timp. Dacă timpul nu este setat, va continua să se rotească. Valorile admisibile pentru viteză: 0-100.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass

rocky.turn_left(50, 1) # Virează la stânga cu viteza 50 timp de o secundă
rocky.turn_left(100) # Virează la stânga cu viteza maximă 100
```

turn_right_by_degree(angle)	Face ca *Rocky* să vireze la dreapta cu un anumit unghi.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
```

```
    pass

rocky.turn_right_by_degree(90) # Virează la dreapta cu un
unghi de 90 de grade
```

turn_left_by_degree(angle)	Face ca *Rocky* să se întoarcă la stânga cu un anumit unghi.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass

rocky.turn_left_by_degree(90) # Virează la stânga cu un
unghi de 90 de grade
```

drive(left_speed, right_speed)	Setarea vitezei celor două roți ale lui *Rocky* în același timp.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass

rocky.drive(50, 50) # Setează viteza ambelor roți la 50
```

stop()	Oprește mișcarea lui *Rocky*

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass

rocky.drive(50, 50) # Setează viteza ambelor roți la 50
rocky.backward(100) # Merge înapoi cu viteza maximă 100
rocky.stop() # Oprește mișcarea
```

Utilizarea senzorilor din dotarea lui Rocky

În tabelul 7 sunt incluse funcțiile *Python* ce permit utilizarea senzorilor și a luminii lui *Rocky*.

Tabelul 7: Funcții *Python* pentru senzorii și lumina lui *Rocky*

<table>
<tr><th>Funcții</th><th>Acțiune</th></tr>
<tr><td>color_ir_sensor.set_led_color(color)</td><td>Setează culoarea LED-ului Rocky. Culoarea poate fi una dintre: „roșu”, „verde”, „albastru”, „galben”, „violet”, „cyan”, „alb”, „negru”. Setarea culorii la „negru” va stinge LED-ul Rocky.</td></tr>
<tr><td colspan="2">Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
rocky.color_ir_sensor.set_led("red")
rocky.color_ir_sensor.set_led("black")
```

</td></tr>
<tr><td>color_ir_sensor.is_obstacle_ahead()</td><td>Spune dacă există obstacol în față. Returnează True sau False. (Setul de senzori Rocky trebuie să fie orientat înainte).</td></tr>
<tr><td colspan="2">Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass
rocky.color_ir_sensor.set_led("red")
rocky.color_ir_sensor.set_led("black")
print(rocky.color_ir_sensor.is_obstacle_ahead())
```

</td></tr>
</table>

color_ir_sensor	Întoarce valoarea matricei de senzori de bază. Poate întoarce valori de la 0-100.
color_ir_sensor.get_light_strength()	Întoarce intensitatea luminii ambientale a mediului. Poate întoarce valori de la 0-100.
color_ir_sensor.get_reflected_light()	Întoarce valoarea luminii de reflecție a suprafeței. Poate întoarce valori de la 0-100.
color_ir_sensor.get_reflected_infrared()	Întoarce valoarea de reflecție *IR* a suprafeței. Poate întoarce valori de la 0-100.
color_ir_sensor.get_greyness()	Întoarce gri prin senzorul de lumină. Poate întoarce valori de la 0-100.
color_ir_sensor.get_red()	Întoarce valoarea roșie. Poate întoarce valori de la 0-255.
color_ir_sensor.get_green()	Întoarce valoarea verde. Poate întoarce valori de la 0-255.
color_ir_sensor.get_blue()	Întoarce valoarea albastră. Poate întoarce valori de la 0-255.
color_ir_sensor.is_color(color_str)	Spune dacă senzorul de culoare detectează culoarea specificată; returnează *True* sau *False*. Numele culorii poate fi unul dintre „roșu”, „verde”, „albastru”, „galben”, „purpuriu”, „cian”, „alb”, „negru”.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
```

```
    pass

print("Luminozitatea și Reflecția: ",
color_ir_sensor.get_light_strength(),
color_ir_sensor.get_reflected_light())
print("Valoarea luminii IR: ",
color_ir_sensor.get_reflected_infrared())
print("Cenușiu: ", color_ir_sensor.get_greyness())
print("Valorile RGB: ", color_ir_sensor.get_red(),
color_ir_sensor.get_green(), color_ir_sensor.get_blue())
print("Este o suprafață roșie? ",
color_ir_sensor.is_color("red"))
```

Comunicarea cu semnalele infraroșu

Codey poate trimite și primi semnale în infraroșu. În tabelul de mai jos sunt examinate codurile *Python* care realizează trimiterea și primirea semnalelor în infraroșu (tabelul 8).

Tabelul 8: Semnalele *IR*

Funcții	Acțiune
ir_send(message)	Trimite un mesaj textual cu Emițătorul Infraroșu *Codey*. Un alt *Codey* poate primi acest mesaj cu Receptorul Infraroșu.

Exemplu:
`import codey, rocky, time, event`
`@event.start`
`def on_start():`
`    pass`
 	
`codey.ir_send("A")`	

ir_receive()	Când se primește o bucată de mesaj trimisă cu semnale în infraroșu, returnează o valoare de tip șir de caractere.

Exemplu:

```
import codey, rocky, time, event
@event.start
def on_start():
    pass

codey.ir_send("A")
codey.show(codey.ir_receive())
```

Referința Api Python. API-ul Python pentru Codey

În această secțiune vor fi examinate următoarele aspecte:

- *API-ul Python pentru Codey* - se referă la unele API-uri pentru driverul integrat al *Codey Rocky*.
- *API Python pentru Rocky* - API pe *Codey*, folosit pentru controlul lui *Rocky* pentru a muta sau pentru a primi datele trimise de la senzori pe *Rocky*.
- *API Python pentru biblioteci terțe* - o interfață integrată pentru biblioteci terțe din *Codey Rocky*, cum ar fi *mqtt* sau *urequest*.
- *API Python pentru extensia modulelor Neuron* - API care poate fi utilizat atunci când adăugăm modulele *Neuron* la *Codey Rocky*.

Led-ul RGB

În tabelul 9 sunt incluse funcțiile *Python* ce permit utilizarea ledului *RGB*:

Tabelul 9: Funcțiile *Python* pentru ledul *RGB*

Funcția	Acțiune
led.show(r, g, b)	Setarea culorii afișate a LED-ului RGB, parametrii: • *r* se referă la valoarea componentei roșii, intervalul de parametri este 0 ~ 255, 0 fără componentă roșie și 255 cea mai mare componentă roșie. • *g* se referă la valoarea componentei verzi, intervalul de parametri este 0 ~ 255, 0 fără componentă verde și 255 cea mai mare componentă verde. • *b* se referă la valoarea componentei albastre, intervalul de parametri este 0 ~ 255, 0 fără componentă albastră și 255 cea mai mare componentă albastră.
led.set_red(val)	Setarea valorii culorii roșii a LED-ului RGB, parametrul:

	• *val* se referă la valoarea componentei roșii, intervalul parametrilor este 0 ~ 255, 0 fără componentă roșie și 255 cea mai mare componentă roșie.
led.set_green(val)	Setarea valorii culorii verzi a LED-ului RGB, parametrul: • *val* se referă la valoarea componentei verzi, intervalul parametrilor este 0 ~ 255, 0 fără componentă verde și 255 cea mai mare componentă verde.
led.set_blue(val)	Setarea valorii culorii albastre a LED-ului RGB, parametrul: • *val* se referă la valoarea componentei albastre, intervalul parametrilor este 0 ~ 255, 0 fără componentă albastră și 255 cea mai mare componentă albastră.
led.off()	Oprirea LED-ului RGB.

Exemplu:

```
import codey
import time

codey.led.show(255,255,255)
time.sleep(2)
codey.led.off()
time.sleep(2)
while True:
    codey.led.set_red(255)
    time.sleep(1)
    codey.led.set_green(255)
    time.sleep(1)
    codey.led.set_blue(255)
    time.sleep(1)
    codey.led.off()
    time.sleep(1)
```

Panoul frontal – display-ul lui Codey

Figura 4 ilustrează grafic panoul frontal al lui *Codey*.

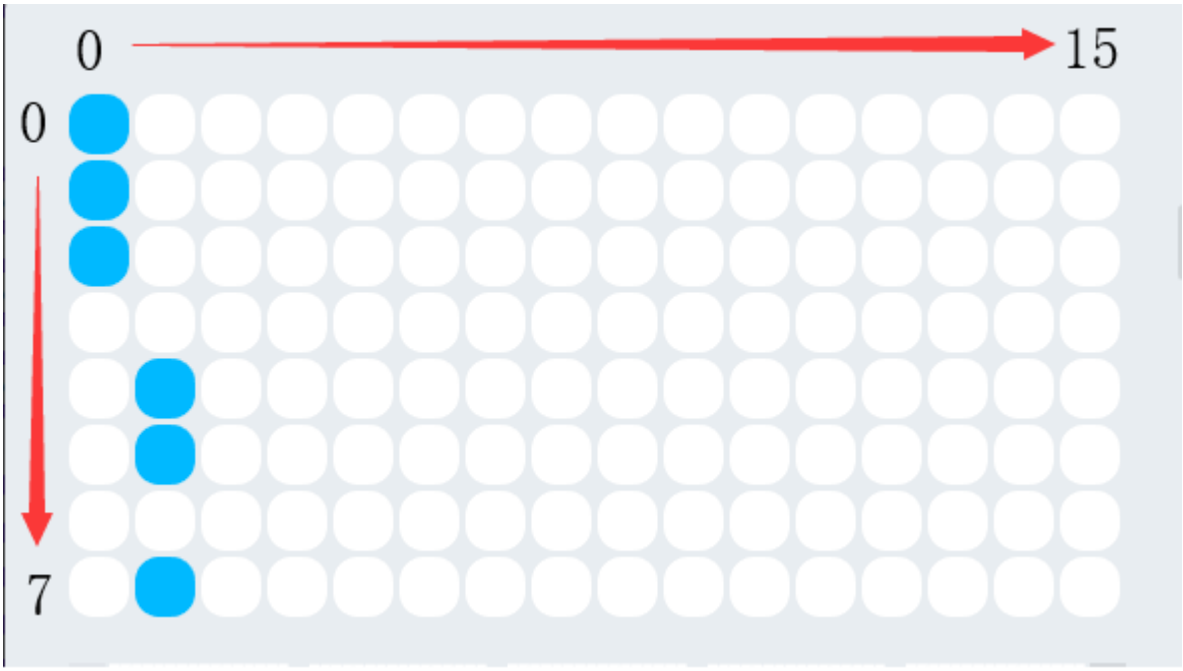


Figura 4: Panoul frontal Codey

După cum se arată în figura de mai sus, panoul frontal are colțul din stânga sus ca punct de coordonată 0, iar direcția *x* și *y* este indicată de săgeți. Despre parametrii afișați, luați ca exemplu figura de mai sus. Cele trei date superioare ale primei coloane sunt aprinse și datele sunt convertite la 11100000, adică hexazecimal 0xe0. Versul celei de-a doua coloane este convertit la 00001101, adică În hexazecimal 0x0d. Toate rețelele din figura de mai sus sunt convertite în e00d0000000000000000000000000000.

În tabelul 10 avem enumerate funcțiile *Python* ce permit utilizarea panoului frontal *Codey*.

Tabelul 10: Funcțiile *Python* pentru panoul frontal al lui *Codey*

Funcția	Acțiune
display.show_image(image, pos_x = 0, pos_y = 0, time_s = None)	Afișarea matricei grafice de puncte personalizată ca parametri de imagine, parametrii: • *image* – șiruri de date, fiecare coloană a matricei de puncte are 8 puncte de afișare, adică 1 octet de date, convertite într-un șir hexazecimal. Prin urmare, 16 coloane de rețele trebuie să fie reprezentate prin 32 șiruri de date. • *pos_x* - afișează pe panou decalajul axei x a graficului. Intervalul de parametri este -15 ~ 15, începe de la poziția 0 ca implicit dacă acest parametru nu este setat. • *pos_y* - afișează decalajul graficului pe axa y a panoului. Intervalul de parametri este -7 ~ 7, începe de la poziția 0 ca implicit dacă acest parametru nu este setat. • *time_s* - afișează timpul în secunde (în secunde). Dacă acest parametru nu este setat, afișajul rămâne neschimbat până când apare un ecran clar sau resetează funcționarea panoului.
display.show(var, pos_x = 0, pos_y = 0, wait = True)	Afișează datele în parametri de date de tip complet, parametrii:

	• *var* - tip complet, unde afișarea tipurilor numerice și de timp este tratată special, iar afișarea formatului de timp trebuie să satisfacă formatul: [x]x: [x]x (expresie regulată d?d: dd?) • *pos_x* - afișează compensarea datelor pe axa x a panoului. Intervalul de parametri este -15 ~ 15. Începe din poziția 0 ca implicit dacă acest parametru nu este setat. • *pos_y* - afișează compensarea datelor pe axa y a panoului. Intervalul de parametri este -7 ~ 7. Începe din poziția 0 ca implicit dacă acest parametru nu este setat. • *wait* - stabilește dacă se blochează afișajul, unde *True*: înseamnă blocare până când afișajul este complet, *False* - înseamnă afișare, dar nu blocare.
display.set_pixel(pos_x, pos_y, status)	Setează luminozitatea și starea de dezactivare a unui singur pixel al panoului, parametrii: • *pos_x* - coordonatele axei x pentru pixelul de pe panou. Intervalul admisibil este 0 ~ 15. • *pos_y* - coordonatele axei y pentru pixelul de pe panou. Intervalul admisibil este 0 ~ 7.

	• *status* - valoare booleană, unde *True* indică faptul că pixelul este aprins și *False* indică faptul că pixelul este dezactivat.
display.get_pixel(pos_x, pos_y)	Întoarce stările actuale de pornire și oprire ale unui singur pixel de pe panou. Valoarea returnată este o valoare booleană, unde *True*: indică faptul că pixelul este aprins și *False*: indică faptul că pixelul este oprit, parametrii: • *pos_x* - coordonatele axei x pentru pixelul de pe panou. Intervalul admisibil este 0 ~ 15. • *pos_y* - coordonatele axei y pentru pixelul de pe panou. Intervalul admisibil este 0 ~ 7.
display.toggle_pixel(pos_x, pos_y)	Comutează stările curente de pornire și oprire ale unui singur pixel de pe panou, parametrii: • *pos_x* - coordonatele axei x pentru pixelul de pe panou. Intervalul admisibil este 0 ~ 15. • *pos_y* - coordonatele axei y pentru pixelul de pe panou. Intervalul admisibil este 0 ~ 7.
display.clear()	Oprește toate luminile LED-ului de pe panoul frontal.

Exemplu:
`import codey`
`import time`	

```
codey.display.show("ffffff")
codey.display.show("123")
time.sleep(1)
codey.display.show("12345", 3, 1)
codey.display.set_pixel(1, 1, True)
image = "fffffffff000000000000000000000000"
codey.display.show_image(image, pos_x = 3, pos_y = 4)
time.sleep(1)
codey.display.clear()
print("[1, 1]:", codey.display.get_pixel(1, 1))
codey.display.show("12:28")
while True:
    codey.display.toggle_pixel(7, 2)
    codey.display.toggle_pixel(7, 4)
    time.sleep(1)
```

Categoria Speaker

Tabelul 11 conține funcțiile *Python* ce permit utilizarea în program a efectelor de sunete:

Tabelul 11: Funcțiile *Python* pentru *Speaker*

Funcția	Acțiune
speaker.stop_sounds()	Stoparea tuturor sunetelor
speaker.play_melody(file_name)	Redarea unui fișier audio, funcția nu se va bloca la redare, dar dacă este apelată continuu, următoarea redare va opri redarea anterioară, parametrii: • *file_name* – de tipul string, numele fișierului audio în format .wav înregistrat în blițul Codey Rocky. Putem introduce sufixul de format .wav sau poate fi omis. Fișierul audio opțional poate fi: • hello.wav : hello • hi.wav : hi • bye.wav : bye • yeah.wav : yeah • wow.wav : wow • laugh.wav : laugh • hum.wav : hum • sad.wav : sad • sigh.wav : sigh • annoyed.wav : annoyed • angry.wav : angry • surprised.wav : scared • yummy.wav : pettish • curious.wav : curious • embarrassed.wav : embarrassed • ready.wav : ready • sprint.wav : sprint • sleepy.wav : snore • meow.wav : meow • start.wav : start • switch.wav : switch

	• beeps.wav : beeps • buzzing.wav : buzz • exhaust.wav : air-out • explosion.wav : explosion • gotcha.wav : gotcha • hurt.wav : painful • jump.wav : jump • laser.wav : laser • level up.wav : level-up • low energy.wav : low-energy • metal clash.wav : metal-clash • prompt tone.wav : prompt-tone • right.wav : right • wrong.wav : wrong • ring.wav : ringtone • score.wav : score • shot.wav : shot • step_1.wav : step_1 • step_2.wav : step_2 • wake.wav : activate • warning.wav : warning
speaker.play_melody_until_done (file_name)	Fișierul audio este redat până când se oprește și funcția blochează redarea, adică următoarea instrucțiune nu poate fi executată până când sunetul este redat, parametrul: • *file_name* – de tip string, numele fișierului audio în formatul wav înregistrat în blițul Codey Rocky.
speaker.play_note(note_num, beat = None)	Redă nota muzicală, definițiile notelor digitale pot fi consultate la sursa: https://en.scratch-wiki.info/wiki/Play_Note_()_for_()_Beats_(block) Parametrii: • *note_num* - valoare numerică, interval de valori 48 - 72 sau tip șir, cum ar fi C4. • *beat* - indică numărul de bătăi, valoarea implicită este redată întotdeauna. Notele și frecvența acestora sunt următoarele:

	• ['C2','65'], ['D2','73'], • ['E2','82'], ['F2','87'], • ['G2','98'], ['A2','110'], • ['B2','123'], ['C3','131'], • ['D3','147'], ['E3','165'], • ['F3','175'], ['G3','196'], • ['A3','220'], ['B3','247'], • ['C4','262'], ['D4','294'], • ['E4','330'], ['F4','349'], • ['G4','392'], ['A4','440'], • ['B4','494'], ['C5','523'], • ['D5','587'], ['E5','659'], • ['F5','698'], ['G5','784'], • ['A5','880'], ['B5','988'], • ['C6','1047'], ['D6','1175'], • ['E6','1319'], ['F6','1397'], • ['G6','1568'], ['A6','1760'], • ['B6','1976'], ['C7','2093'], • ['D7','2349'], ['E7','2637'], • ['F7','2794'], ['G7','3136'], • ['A7','3520'], ['B7','3951'], • ['C8','4186'], ['D8','4699'],
speaker.play_tone(frequency, time = None)	Redă frecvența sunetului și timpul de redare a acestuia, parametrii: • *frequence* - date numerice, care indică frecvența sunetului care este redat și intervalul de valori al acestuia este de 0 ~ 5000. • *time* - date numerice, care indică timpul de redare (în milisecunde - ms) și intervalul său de valori este 0 ~ limita intervalului de valori.
speaker.rest(number)	Oprește ritmul, parametrul: • *number* - date numerice, ce indică numărul de bătăi întrerupte, intervalul său de valori este 0 ~ limita intervalului de valori.
speaker.volume	Generează date numerice, care indică valoarea proprietății volumului. Putem

	modifica această valoare pentru a controla volumul sau o putem citi această valoare. Intervalul său de valori este 0 ~ 100.
speaker.tempo	Generează date numerice, care indică natura vitezei de redare, în bătăi pe minut, care este lungimea fiecărei bătăi. Intervalul admisibil de valori este 6 ~ 600. Valoarea implicită este 60, ceea ce înseamnă că durata unei bătăi este 1 secunda. Bătăile funcțiilor *rest* și *play_note* sunt afectate de această constantă.

Exemplu:

```
import codey
import time

codey.speaker.play_melody("hello", True)
codey.display.show("hello")
codey.display.clear()

codey.speaker.play_note(48, 1)
codey.speaker.rest(1)
codey.display.show("note")
codey.display.clear()
codey.speaker.play_note("C4", 1)
codey.speaker.rest(1)
codey.display.show("C4")
codey.display.clear()
codey.speaker.play_tone(1000, 2)
codey.speaker.rest(1)
codey.display.show("tone")
codey.display.clear()
print("tempo:", end = "")
print(codey.speaker.tempo)
codey.speaker.play_note("C4", 1)
codey.speaker.rest(1)
codey.speaker.tempo = 120
codey.speaker.volume = 20
codey.speaker.play_note("C4", 1)
codey.speaker.rest(1)
```

Senzorii de sunet și lumină, Potențiometrul, Butonul A, Butonul B și Butonul C

Funcțiile ce permit utilizarea senzorilor lui *Codey Rocky* sunt expuse în tabelul 12.

Tabelul 12: Funcțiile *Python* pentru senzorii de sunet și lumină, Potențiometrul, Butonul A, Butonul B și Butonul C

Funcția	Acțiune
sound_sensor.get_loudness()	Returnează intensitatea sunetului detectată de senzorul de sunet, iar valoarea returnată este volumul. Intervalul de valori admisibile este 0 ~ 100.

Exemplu:
`import codey`
 	
`while True:`
`    codey.display.show(codey.sound_sensor.get_loudness())`	

light_sensor.get_value()	Returnează intensitatea luminii detectată de senzorul de lumină, iar valoarea de revenire este valoarea intensității luminii vizibile. Intervalul de valori admisibile este 0 ~ 100.

Exemplu:
`import codey`
 	
`while True:`
`    codey.display.show(codey.light_sensor.get_value())`	

potentiometer.get_value()	Returnează valoarea curentă a butonului potențiometrului. Intervalul de valori este 0 ~ 100.

Exemplu:
`import codey`
 	
`while True:`
`    codey.display.show(codey.potentiometer.get_value())`	

button_a.is_pressed()	Returnează starea curentă a butonului A. Rezultatul returnat este *True* dacă butonul este apăsat sau *False* dacă butonul nu este apăsat.

Exemplu:

```
import codey

def loop():
    while True:
        if codey.button_a.is_pressed():
            print("button A is pressed")
loop()
```

button_b.is_pressed()	Returnează starea curentă a butonului B. Rezultatul returnat este *True* dacă butonul este apăsat sau *False* dacă butonul nu este apăsat.

Exemplu:

```
import codey

def loop():
    while True:
        if codey.button_b.is_pressed():
            print("button B is pressed")
loop()
```

button_c.is_pressed()	Returnează starea curentă a butonului C. Rezultatul returnat este *True* dacă butonul este apăsat sau *False* dacă butonul nu este apăsat.

Exemplu:

```
import codey

def loop():
    while True:
        if codey.button_c.is_pressed():
            print("button C is pressed")
loop()
```

Motion_sensor - senzorul de mișcare

După cum se arată în imaginea de mai jos (fig.5), direcția rulării și a pasului se bazează pe regula șurubului de dreapta. Atât direcția rulării, cât și a pasului sunt de 0 ° când *Codey* este așezat orizontal.

Intervalul direcției de rulare este: -90 ° ~ 90 °

Intervalul direcției pasului: -180 ° ~ 180 °

Figura 5: Direcții de mișcare

În tabelul 13 sunt indicate funcțiile *Python* ce permit utilizarea senzorului de mișcare.

Tabelul 13: Funcțiile *Python* pentru senzorul de mișcare

Funcția	Acțiune
motion_sensor.get_roll()	Obținem unghiul *Euler* al direcției de rulare, intervalul de date returnat este de -90 ~ 90.
motion_sensor.get_pitch()	Obținem pasul unghiului *Euler*, intervalul de date care pot fi returnate este -180 ~ 180.
motion_sensor.get_yaw()	Obținem valoarea unghiului *Euler*, intervalul de date returnat este 0 ~ 360. Deoarece senzorul *Codey* de la bord este un senzor cu șase axe, nu există busolă

	electronică. Deci, de fapt, unghiul de girație este doar integralul vitezei unghiulare a axei Z. Dacă dorim să obținem un unghi adevărat de girație, acest API nu este potrivit pentru utilizare.
motion_sensor.get_rotation(axis)	Obținem unghiul la care se rotește *Codey* pe cele trei axe, iar direcția în sens invers acelor de ceasornic este direcția pozitivă, parametrul: • *axis* – de tip *String*, notat cu x, y, z reprezentând axa definită de *Codey*.
motion_sensor.reset_rotation(axis = "all")	Întoarce unghiul curent de rotație inițială în jurul celor trei axe este 0, iar *get_rotation ()* va începe la 0, parametrul: • *axis* – de tip String notat cu x, y, z reprezentând axa definită de Codey și • *all* - reprezentând toate cele trei axe. Aceasta este și valoarea implicită pentru această funcție.
motion_sensor.is_shaked()	Verifică dacă *Codey* este zdruncinat, valoarea returnată este o valoare booleană, unde *True* înseamnă că *Codey* este zdruncinat și *False* înseamnă că *Codey* nu este scuturat.
motion_sensor.get_shake_strength()	Dacă *Codey* este zdruncinat, această funcție poate obține intensitatea zdruncinării. Valoarea intervalului valorii de retur este de 0 ~ 100. Cu cât este mai mare valoarea, cu atât este mai mare intensitatea zdruncinării.
motion_sensor.is_tilted_left()	Verifică dacă *Codey* este înclinat spre stânga, iar valoarea returnată este o valoare booleană, unde *True* înseamnă că *Codey* este înclinat spre stânga și *False* înseamnă că Codey nu este înclinat spre stânga.
motion_sensor.is_tilted_right()	Verifică dacă *Codey* este înclinat spre dreapta, iar valoarea returnată este o valoare booleană, unde *True* înseamnă că *Codey* este înclinat spre dreapta și

	False înseamnă că *Codey* nu este înclinat spre dreapta.
motion_sensor.is_ears_up()	Verifică dacă urechea lui *Codey* este orientată în sus, valoarea returnată este o valoare booleană, unde *True* înseamnă că urechea lui *Codey* este orientată în sus și *False* înseamnă că urechea Codey nu este orientată în sus.
motion_sensor.is_ears_down()	Verifică dacă urechea lui *Codey* este orientată în jos, valoarea returnată este o valoare booleană, unde *True* înseamnă că urechea lui *Codey* este orientată în jos și *False* înseamnă că urechea lui *Codey* nu este orientată în jos.
motion_sensor.is_display_up()	Verifică dacă panoul frontal al lui *Codey* este orientat în sus, valoarea returnată este o valoare booleană, unde *True* înseamnă că panoul *Codey* este orientat în sus și *False* înseamnă că panoul *Codey* nu este orientat în sus.
motion_sensor.is_display_down()	Verifică dacă panoul *Codey* este orientat în jos, valoarea returnată este o valoare booleană, unde *True* înseamnă că panoul *Codey* este orientat în jos și *False* înseamnă că panoul *Codey* nu este orientat în jos.
motion_sensor.is_upright()	Verifică dacă *Codey* este poziționat vertical, valoarea returnată este o valoare booleană, unde *True* înseamnă că *Codey* este poziționat vertical și *False* - Codey nu este poziționat *vertical*.
motion_sensor.get_acceleration(axis)	Obținem valorile de accelerație ale celor trei axe în m / s ^ 2, parametrul: • *axis* – de tip string, notat cu *x, y, z* reprezentând axa definită de *Codey*.
motion_sensor.get_gyroscope(axis)	Obținem valorile vitezei unghiulare ale celor trei axe în ° / sec, parametrul: • *axis* – de tip string, notat cu *x, y, z* reprezentând axa definită de *Codey*.

Exemplul 1:
`import codey`
`import time`	

```
while True:
    roll = codey.motion_sensor.get_roll()
    pitch = codey.motion_sensor.get_pitch()
    yaw = codey.motion_sensor.get_yaw()
    print("roll:", end = "")
    print(roll, end = "")
    print("   ,pitch:", end = "")
    print(pitch, end = "")
    print("   ,yaw:", end = "")
    print(yaw)
    time.sleep(0.05)
```

Exemplul 2:

```
import codey

while True:
    if codey.motion_sensor.is_shaked():
        print("shake_strength:", end = "")
        print(codey.motion_sensor.get_shake_strength())
```

Exemplul 3:

```
import codey

while True:
    if codey.motion_sensor.is_tilted_left():
        print("tilted_left")
    if codey.motion_sensor.is_tilted_right():
        print("tilted_right")
    if codey.motion_sensor.is_ears_up():
        print("ears_up")
    if codey.motion_sensor.is_ears_down():
        print("ears_down")
    if codey.motion_sensor.is_display_up():
        print("display_up")
    if codey.motion_sensor.is_display_down():
        print("display_down")
    if codey.motion_sensor.is_upright():
        print("upright")
```

Transmițătorul IR

Tabelul 14 conține funcțiile *Python* ce permit să utilizăm transmițătorul IR în programarea lui *Codey*.

Tabelul 14: Funcțiile *Python* pentru transmițătorul IR

Funcția	Acțiune
ir.receive()	Returnează informațiile șirului primite de receptorul infraroșu, astfel încât datele trimise de transmițătorul infraroșu trebuie să se termine cu \ *n*. Dacă este o comandă de control de la distanță care primește protocolul de codare NEC, se utilizează o altă funcție *receive_remote_code ()*.
ir.receive_remote_code()	Obținem date de la telecomanda cu infraroșu. Datele conțin două părți: adresă și conținut, deci returnează o listă de date cu lungimea 2. Primul parametru este codul de adresă, iar ultimul parametru este codul de date.
ir.send(str)	Trimite un șir de caractere infraroșu, parametrul: • *str* - date de tip string care vor fi emise, funcția de trimitere va adăuga automat terminația \n la sfârșitul șirului.
ir.start_learning()	Pornește învățarea în infraroșu și acceptă doar comenzile de control de la distanță care învață protocolul standard NEC.
ir.stop_learning()	Oprește învățarea în infraroșu.
ir.save_learned_result(index)	Salvează rezultatul de codificare în infraroșu în zona corespunzătoare, parametrul: • *index* - gama de valori a indexului este 0 ~ 15, în total 16 zone de stocare.
ir.send_learned_result(index = 1)	Trimite codul infraroșu salvat pentru învățarea în infraroșu, rezultatul învățării zonei cu index = 1 este setat ca implicit, parametrul: • *index* - intervalul de valori a indexului este 0 ~ 15, în total 16 zone de stocare.

ir.learn(time = 3)	Va returna timpul de învățare în infraroșu, după ce am apelat acest API, va salva informațiile în infraroșu aflate în secunde de timp. Implicit este salvat în zona cu *index = 1*, parametru: • *time* - timpul de învățare, în secunde.

Exemplul 1:

```
import codey
import event

@event.start
def start_cb():
    print("start event succeeded")
    while True:

codey.display.show(codey.ir.receive_remote_code()[1])
```

Exemplul 2:

```
import codey
import event

@event.button_a_pressed
def button_a_cb():
    print("button a event succeeded")
    codey.ir.learn()
    codey.led.show(0, 100, 0)

@event.button_b_pressed
def button_a_cb():
    print("button b event succeeded")
    while True:
        codey.ir.send_learned_result()

@event.button_c_pressed
def button_c_cb():
    print("button c event succeeded")
    while True:
        codey.display.show(codey.ir.receive())
```

WI-FI

Tabelul 15 conține funcțiile *Python* utilizate pentru WI-FI.

Tabelul 15: Funcțiile *Python* pentru WI-FI

Funcția Pyton	Acțiune
wifi.start(ssid = "wifi_ssid", password = "password", mode = codey.wifi.STA)	Pornește conexiunea Wi-Fi, API-ul nu va bloca procesul, ieșirea API nu înseamnă că Wi-Fi este conectat, trebuie să apelăm funcția *wifi.is_connected ()* pentru a conecta. Parametrii: • *ssid* - de tip șir de caractere, ce indică contul Wi-Fi. • *password* – de tip șir, ce indică parola Wi-Fi. • *mode* - pornește modul Wi-Fi.
wifi.is_connected()	Verifică dacă este conectat Wi-Fi, valoarea returnată este booleană, unde *True* înseamnă că Wi-Fi a stabilit o conexiune, *False* înseamnă că Wi-Fi nu a stabilit încă o conexiune.
wifi.STA -constantă	Modul stației Wi-Fi, adică modul adaptor wireless. În acest mod, Wi-Fi poate fi conectat la router.
wifi.AP - constantă	Modul de acces fără fir, rutarea generală fără fir funcționează în acest mod. În acest mod, permite accesul altor dispozitive fără fir.
wifi.APSTA - constantă	Modurile Wi-Fi AP și STA coexistă.

Exemplu:

```
import codey
codey.wifi.start('Vascan', '079010567', codey.wifi.STA)
codey.led.show(0,0,0)
while True:
    if codey.wifi.is_connected():
        codey.led.show(0,0,255)

    else:
        codey.led.show(0,0,0)
```

Battery – Baterie de litiu încorporată

Tabelul 16 conține funcții *Python* atașate la bateria de litiu încorporată

Tabelul 16: Funcțiile *Python* pentru bateria de litiu încorporată

Funcția	Acțiune
battery.get_voltage()	Obținem tensiunea curentă a bateriei, valoarea de retur este o dată în virgulă mobilă. Unitatea de măsură este *V* (volți).
battery.get_percentage()	Obținem procentul de energie rămasă a bateriei. Valoarea returnată este un număr întreg. Intervalul de date este de la 0 la 100, unde 100 înseamnă că există încă 100% din bateria rămasă.

Exemplu:
`import codey`
 	
`while True:`
`  print("vol" + str(codey.battery.get_voltage()))`
`  print("percentage" + str(codey.battery.get_percentage()))`	

codey_timer - Temporizatorul

Funcțiile destinate programării evenimentelor ce țin de timer-ul lui *Codey* sunt incluse în tabelul 17.

Tabelul 17: Funcțiile *Python* pentru *codey_timer*

Funcția Pyton	Acțiune
codey.get_timer()	Obținem valoarea curentă a temporizatorului (temporizatorul rulează din momentul în care scriptul utilizatorului pornește), valoarea returnată este o dată în virgulă mobilă. Unitatea de măsură- secunde.
codey.reset_timer()	Inițializează valoarea temporizatorului.

Exemplu:
`import codey`
 	
`codey.reset_timer()`
 	
`while True:`
`  print("time:", end = "")`
`  print(codey.get_timer())`	

codey_broadcast - Modul de difuzare

În tabelul 18 se conține funcția *Python* ce permite utilizarea modului de difuzare Codey.

Tabelul 18: Funcțiile *Python* pentru modul de difuzare *Codey*

Funcția	Acțiune
codey.broadcast(str)	O transmisie poate fi trimisă la portul serial, *Bluetooth* și la propria unitate de monitorizare a evenimentelor.Pparametrul: • *str* – va exprima conținutul difuzării.
Exemplu: `import codey`	

```
import event

@event.button_a_pressed
def button_a_cb():
    print("button a event succeeded")
    codey.broadcast("salut")

@event.received("salut")
def received_cb():
    print("received message: salut")import codey
import event

@event.button_a_pressed
def button_a_cb():
    print("button a event succeeded")
    codey.broadcast("Salut")

@event.received("salut")
def received_cb():
    print("received message: salut")
```

codey_external_module_detect - Detectarea modulelor de acces extern

Tabelul 19 conține funcții *Python* de detectare a modulelor de acces extern.

Tabelul 19: Funcțiile *Python* pentru modulele de acces extern *Codey*

Funcția	Acțiune
codey.has_neuron_connected()	Verifică dacă există vreun modul de neuroni conectat la *Codey*, iar valoarea returnată este o valoare booleană, unde *True* indică faptul că modulul de neuroni este conectat la *Codey* (inclusiv conectarea *Rocky*) și *False* indică faptul că nu există niciun modul de neuroni conectat la *Codey*.
codey.is_rocky_connected()	Verifică dacă *Rocky* este conectat la *Codey*, valoarea returnată este o valoare booleană, unde *True* înseamnă că există un *Rocky* conectat la *Codey*, iar *False* înseamnă că nu există *Rocky* conectat la *Codey*.

Exemplu:

```
import codey
import time

while True:
    if codey.is_rocky_connected():
        print("rocky is in")
    else:
        print("rocky is out")
    time.sleep(1)
```

codey_script_control - Controlul script-ului

Tabelul 20 conține funționalitățile și funcțiile modulului *codey_script_control*.

Tabelul 20: Funcțiile *Python* pentru *codey_script_control*

Funcția	Acțiune
codey.stop_this_script()	Opriți scriptul curent, în concordanță cu caracteristica scriptului de oprire *Scratch*.
codey.stop_other_scripts()	Oprește alte scripturi, în concordanță cu caracteristica *Scratch* ce oprește alte scripturi.
codey.stop_this_script()	Oprește toate scripturile, în concordanță cu caracteristica *Scratch* ce oprește toate scripturile.

Exemplu:

```
import codey
import time
import event

@event.start
def start_cb():
    while True:
        print("start cb executing...")
        time.sleep(1)
        print("oprește acest script")
        codey.stop_this_script()
@event.button_a_pressed
def button_a_cb():
    codey.stop_other_scripts()
    while True:
        print("button a event")
@event.button_b_pressed
def button_b_cb():
    codey.stop_other_scripts()
    while True:
        print("button b event")
@event.button_c_pressed
def button_c_cb():
    codey.stop_all_scripts()
   print("oprește toate script-urile")
```

event - Unitatea de procesare a evenimentelor

Există două moduri de înscriere a evenimentelor: unul este înregistrarea și altul – utilizarea unui decorator.

Exemplul 1 (înregistrarea):

```
import codey
import time
import event

def start_cb():
    while True:
        codey.led.show(255, 0, 0)
        time.sleep(1)
        codey.led.show(0, 0, 0)
        time.sleep(1)
event.start(start_cb)
```

Exemplul 2 (utilizarea unui decorator):

```
import codey
import time
import event

@event.start

def start_callback():
    while True:
        codey.led.show(255, 0, 0)
        time.sleep(1)
        codey.led.show(0, 0, 0)
        time.sleep(1)
```

Tabelul 21 conține funcții *Python* de procesare a evenimentelor.

Tabelul 21: Funcțiile *Python* pentru unitatea de procesare a evenimentelor

Funcția	Acțiune
event.start(callback)	Evenimentul de pornire.
event.shaked(callback)	Evenimentul *Codey* a fost zdruncinat.

event.received(callback, msgstr)	Eveniment de detectare a recepției prin difuzare. În plus față de parametrul de apel invers (callback), parametrul: • *msgstr* – de tip de șir de caractere, conține șirul care se potrivește. Evenimentul va fi declanșat atunci când șirul primit se potrivește cu șirul care se indică.
event.button_a_pressed(callback)	Evenimentul Butonul A este apăsat.
event.button_b_pressed(callback)	Evenimentul Butonul B este apăsat.
event.button_c_pressed(callback)	Evenimentul Butonul C este apăsat.
event.tilted_left(callback)	Evenimentul *Codey* înclinat la stânga.
event.tilted_right(callback)	Evenimentul *Codey* înclinat la dreapta.
event.ears_up(callback)	Evenimentul *Codey* cu urechile în sus.
event.ears_down(callback)	Evenimentul *Codey* cu urechile în jos.
event.ir_received(callback, ir_str)	Eveniment de detectare a recepției cu infraroșu. În plus față de parametrul de apel invers (callback), parametrul: • *ir_str* – de tip de șir, indică șirul care se potrivește. Evenimentul va fi declanșat atunci când șirul primit se potrivește cu șirul care se indicat.
event.greater_than(callback, threshold, type_str)	Când volumul este peste o anumită valoare, se apelează această funcție. În plus față de parametrul de apel invers (callback), parametrii: • *threshold* - setează valoarea pragului pentru declanșare; • *type_str* - date de tip șir de caractere, în prezent acceptă doar sound_sensor: senzor de volum, timer: timer.
event.less_than(callback, threshold, type_str)	Eveniment de comparație a luminozității față de o anumită valoare, pe lângă parametrul de apel invers (callback). Parametrii: • *threshold*, setează pragul pentru declanșare • *type_str* - șir de date, în prezent acceptă doar light_sensor: senzor de lumină.

Exemplu:
`import codey`
`import event`	

```
@event.button_a_pressed
def button_a_cb():
    print("eveniment declansat de butonul a ")

@event.button_b_pressed
def button_b_cb():
    print("eveniment declansat de butonul b")

@event.button_c_pressed
def button_c_cb():
    print("eveniment declansat de butonul c ")

@event.greater_than(20, "sound_sensor")
def sound_sensor_cb():
    print("evenimentul declansat are sunet prea puternic ")

@event.greater_than(5, "timer")
def timer_cb():
    print("a fost declansat un eveniment cu timer mai mare")

@event.less_than(30, "light_sensor")
def light_sensor_cb():
    print("a fost declansat un eveniment cu luminozitate
prea mare")
```

API Python pentru Rocky. Mișcarea lui Rocky

Tabelul 22 conține funcțiile *Python* destinate mișcării lui *Rocky*

Tabelul 22: Funcțiile *Python* pentru mișcarea lui *Rocky*

Funcția	Acțiune
rocky.stop()	*Rocky* se oprește din mișcare.
rocky.forward(speed, t = None, straight = False)	*Rocky* merge înainte, parametrii: • *speed* – reprezintă valoarea vitezei de mișcare, intervalul parametrilor este -100 ~ 100, numerele negative reprezintă înapoi, numerele pozitive reprezintă înainte. • *t* - reprezintă valoarea timpului de mișcare, în secunde, intervalul de parametri este 0 ~ limita intervalului de valori. Dacă este setat la 1, înseamnă că Rocky se va deplasa înainte timp de 1 s. Dacă acest parametru nu este setat, starea înainte este menținută până când există comanda de oprire a mișcării sau comanda de mișcare nouă. • *straight* - activează senzorul giroscopic pentru a corecta sau nu direcția înainte. Dacă acest parametru nu este setat, nu este activat în mod implicit.
rocky.backward(speed, t = None, straight = False)	*Rocky* se mișcă înapoi, parametrii: • *speed* – reprezintă valoarea vitezei de mișcare, intervalul parametrului este -100 ~ 100, numerele negative reprezintă înainte, numerele pozitive reprezintă înapoi. • *t* -reprezintă valoarea timpului de mișcare, în secunde, intervalul de parametri este 0 ~ limita intervalului de valori. Dacă este setat la 1, înseamnă că *Rocky* se va deplasa înapoi timp de 1 s. Dacă acest parametru nu este setat, starea înapoi

	este menținută până când există o comandă de oprire a mișcării sau o comandă de mișcare nouă. • *straight* - activează senzorul giroscopic pentru a corecta sau nu direcția înapoi. Dacă acest parametru nu este setat, nu este activat în mod implicit.
rocky.turn_left(speed, t = None)	*Rocky* cotește la stânga, parametrii: • *speed* -reprezintă valoarea vitezei de rotație, intervalul de parametrului este -100 ~ 100, numerele negative reprezintă virajul la dreapta, numerele pozitive reprezintă virajul la stânga. • *t* - reprezintă valoarea timpului de mișcare, în secunde, intervalul de parametri este 0 ~ limita intervalului de valori. Dacă este setat la 1, înseamnă că *Rocky* se va întoarce la stânga timp de 1 s. Dacă acest parametru nu este setat, starea de viraj la stânga este menținută până când apare o comandă de oprire a mișcării sau o comandă de mișcare nouă.
rocky.turn_right(speed, t = None)	*Rocky* virează la dreapta, parametrii: • *speed* – reprezintă valoarea vitezei de rotație, intervalul de parametri este -100 ~ 100, numerele negative reprezintă virajul la dreapta, numerele pozitive reprezintă virajul la stânga. • *t* - reprezintă valoarea timpului de mișcare, în secunde, intervalul de parametri este 0 ~ limita intervalului de valori. Dacă este setat la 1, înseamnă că Rocky se va întoarce la stânga timp de 1 s. Dacă acest parametru nu este setat, starea de viraj la stânga este menținută până când apare o comandă de oprire a mișcării sau o comandă de mișcare nouă.

rocky.drive(left_power, right_power)	Rotiri în funcție de valoarea setată pentru fiecare motor, parametrii: • *left_power* – reprezintă viteza motorului roții stângi, intervalul parametrilor este -100 ~ 100, numerele negative reprezintă roata stângă se rotește înapoi, numerele pozitive reprezintă roata stângă se rotește înainte. • *right_power* - reprezintă viteza motorului roții drepte, intervalul parametrilor este -100 ~ 100, numerele negative reprezintă roata dreaptă se rotește înapoi, numerele pozitive reprezintă roata dreaptă se rotește înainte.
rocky.turn_right_by_degree(angle, speed = 40)	*Rocky* cotește la dreapta în funcție de gradele stabilite, parametrii: • *angle* - stabilește unghiul de rotație, numerele negative reprezintă virajul la dreapta, numerele pozitive reprezintă virajul la stânga. • *speed* – reprezintă viteza de virare, intervalul parametrilor este de 0 ~ 100, dacă acest parametru nu este setat, viteza implicită este de 40. (Deoarece senzorul giroscopic este utilizat pentru rotirea unghiului specific, se recomandă să nu modificăm viteza pentru a evita unghiul de virare inexact).
rocky.turn_left_by_degree(angle, speed = 40)	*Rocky* cotește la stânga în funcție de gradele stabilite, parametrii: • *angle* - stabilește unghiul de rotație, numerele negative reprezintă virajul la dreapta, numerele pozitive reprezintă virajul la stânga. • *speed* – reprezintă viteza de virare, intervalul parametrilor este de 0 ~ 100, dacă acest parametru nu este setat, viteza implicită este de 40. (Deoarece senzorul giroscopic este utilizat pentru rotirea unghiului specific, se recomandă să nu

	modificăm viteza pentru a evita unghiul de virare inexact).

Exemplu:

```
import codey
import rocky
import time

rocky.forward(50, 1)
rocky.stop()
rocky.backward(50, 1)
rocky.turn_left(50, 1)
rocky.turn_right(50, 1)
rocky.drive(50, 80)
time.sleep(2)
while True:
    rocky.turn_right_by_degree(80, 40)
    rocky.turn_right_by_degree(80, 20)
```

color_ir_sensor - Senzorul IR de culoare

După cum se arată în figura 6, senzorii din fața lui *Rocky* sunt:

- *LED alb* - lumină albă pentru a detecta intensitatea reflexiei luminii vizibile pe suprafața obiectului cu ajutorul senzorului de lumină vizibilă.
- *Senzor de lumină vizibilă* - detectează intensitatea luminii vizibile.
- *LED RGB* - LED cu valoare specifică RGB pentru a obține recunoașterea culorii cu ajutorul senzorului de lumină vizibilă.
- *Senzorul de lumină cu infraroșu* - detectează intensitatea luminii în infraroșu.
- *Transmițător cu infraroșu* - transmite lumina infraroșie pentru a detecta intensitatea reflexiei luminii infraroșii pe suprafața obiectului cu ajutorul senzorului de lumină infraroșu.

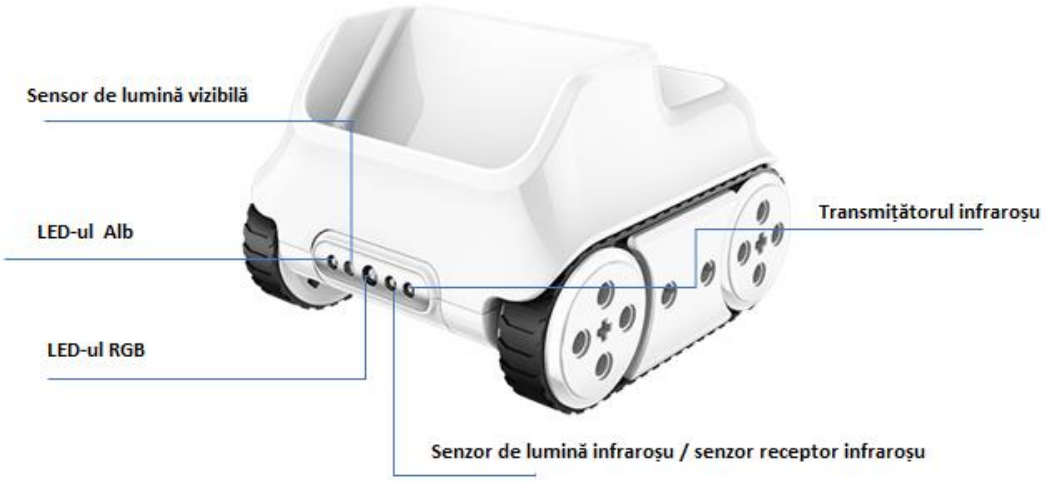


Figura 6: Senzorii din față a lui Rocky

Tabelul 23 conține funcțiile *Python* ce pot fi utilizate în programarea senzorilor de culoare și lumină.

Tabelul 23: Funcțiile *Python* pentru senzorii de culoare și lumină

Funcția	Acțiune
color_ir_sensor.get_red()	Obținem dimensiunea componentei de culoare roșie a senzorului de culoare, intervalul de parametri este 0 ~ 100.
color_ir_sensor.get_green()	Obținem dimensiunea componentei de culoare verde a senzorului de culoare, intervalul de parametri este 0 ~ 100.
color_ir_sensor.get_blue()	Obținem dimensiunea componentei de culoare albastră a senzorului de culoare, intervalul de parametri este 0 ~ 100.
color_ir_sensor.is_color(color_str)	Judecă dacă este detectată o culoare potrivită, parametrul: • *color_str* – reprezintă tipul de culoare, inclusiv roșu, verde, albastru, galben, cian, violet, alb, negru, parametrul corespunzător este roșu,

	verde, albastru, galben, cian, violet, alb, negru. Valoarea returnată este booleană, *True* reprezintă potrivirea culorilor, *False* reprezintă culorile care nu se potrivesc.
color_ir_sensor.get_light_strength()	Obținem intensitatea luminii ambientale detectată de senzorul de lumină vizibilă, intervalul de parametri este 0 ~ 100.
color_ir_sensor.get_greyness()	Obținem valoarea în tonuri de gri detectată de senzorul de lumină vizibilă (folosind LED RGB și senzorul de lumină vizibilă), intervalul parametrilor este de 0 ~ 100.
color_ir_sensor.get_reflected_light()	Obținem intensitatea reflexiei luminii vizibile detectată de senzorul de lumină vizibilă, intervalul parametrilor este de 0 ~ 100.
color_ir_sensor.get_reflected_infrared()	Obținem intensitatea de reflecție a luminii în infraroșu detectată de tubul de recepție a luminii în infraroșu, intervalul parametrilor este de 0 ~ 100.
color_ir_sensor.is_obstacle_ahead()	Detectează dacă există obstacole în față, valoarea de returnare este booleană, *True* reprezintă obstacole, *False* nu reprezintă obstacole.
color_ir_sensor.set_led_color(color_name)	Setează culoarea pentru lumina LED RGB a senzorului de culoare, parametrul: *color_name*, - inclusiv roșu, verde, albastru, galben, cyan, violet, alb, negru, parametrul corespunzător este *red, green, blue, yellow, cyan, purple, white, black*.

Exemplu:

```
import codey
import rocky

while True:
    if rocky.color_ir_sensor.is_obstacle_ahead():
```

```
        rocky.color_ir_sensor.set_led_color('white')
    else:
      rocky.color_ir_sensor.set_led_color('black')
```

API Python pentru biblioteci terțe

urequests - Modul de solicitare a rețelei

Tabelul 24 conține funcții destinate modului de solicitate a rețelei.

Tabelul 24: Funcțiile *Python* pentru modul de utilizare a rețelei

Funcția	Acțiune
urequests.request(method, url, data=None, json=None, headers={})	Trimite o cerere de rețea, aceasta va bloca datele de răspuns returnate în rețea, parametrii: • *method* – reprezintă o metodă de stabilire a unei cereri de rețea. De exemplu. HEAD, GET, POST, PUT, PATCH, DELETE. • *URL* - reprezintă URL-ul solicitării de rețea. • *data* (opțional) - reprezintă un dicționar, o listă de tuple [(cheie, valoare)] (va fi în formă codată), octet sau obiect de fișier de clasă trimis în corpul cererii. • *json* (opțional), date json trimise în corpul cererii. • *headers* (opțional) – reprezintă un dicționar de antete HTTP care trebuie trimis împreună cu cererea.
urequests.head(url, **kw)	Trimite o cerere HEAD, tipul de returnare este răspunsul la cerere, parametrii: • *URL* – este URL-ul rețelei solicitate. • *** kw* - solicită parametri opționali.
urequests.get(url, **kw)	Trimite o cerere GET, tipul de returnare este răspunsul la cerere, parametrii: • *URL* – este URL-ul rețelei solicitate. • *** kw* - solicită parametri opționali.
urequests.post(url, **kw)	Trimite o cerere POST, tipul de returnare este răspunsul la cerere, parametrii:

	• *URL* – este URL-ul rețelei solicitate. • ** *kw* - solicită parametri opționali.
urequests.put(url, **kw)	Trimite o cerere PUT, tipul de returnare este răspunsul la cerere, parametrii: • *URL* – este URL-ul rețelei solicitate. • ** *kw* - solicită parametri opționali.
urequests.patch(url, **kw)	Trimite o cerere PATCH, tipul de returnare este răspunsul la cerere, parametrii: • *URL* – este URL-ul rețelei solicitate. • ** *kw* - solicită parametri opționali.
urequests.delete(url, **kw)	Trimite o cerere DELETE, tipul de returnare este răspunsul la cerere, parametrii: • *URL* – este URL-ul rețelei solicitate. • ** *kw* - solicită parametri opționali.

Exemplul 1:

```
import codey
import urequests as requests
import time

# Completați aici ssid-ul și parola routerului.
codey.wifi.start('wifi_ssid', 'password')
codey.led.show(0,0,0)
while True:
    if codey.wifi.is_connected():
        codey.led.show(0,0,255)
        res = requests.get(url='http://www.baidu.com/')
        print(res.text)
        time.sleep(3)
    else:
        codey.led.show(0,0,0)
```

Exemplul 2:

```
import codey
import urequests as requests
import time

# Completați aici ssid-ul și parola routerului.
codey.wifi.start('wifi_ssid', 'password')
codey.led.show(0,0,0)
hour = minite = second = "00"
while True:
    if codey.wifi.is_connected():
        try:
```

```
            res = requests.get(url =
'http://www.time.ac.cn/timeflash.asp?user=flash').text
            hour_begin = res.find('<hour>') + len('<hour>')
            hour_end = res.find('</hour>')
            minite_begin = res.find('<minite>') +
len('<minite>')
            minite_end = res.find('</minite>')
            second_begin = res.find('<second>') +
len('<second>')
            second_end = res.find('</second>')
            if hour_begin > len('<hour>') and hour_end >
hour_begin and \
                minite_begin > len('<minite>') and minite_end
> minite_begin and \
                second_begin > len('<second>') and second_end
> second_begin:

                if hour_end - hour_begin == 1:
                    hour = '0' + res[hour_begin:hour_end]
                elif hour_end - hour_begin == 2:
                    hour = res[hour_begin:hour_end]

                if minite_end - minite_begin == 1:
                    minite = '0' +
res[minite_begin:minite_end]
                elif minite_end - minite_begin == 2:
                    minite = res[minite_begin:minite_end]

                if second_end - second_begin == 1:
                    second = '0' +
res[second_begin:second_end]
                elif second_end - second_begin == 2:
                    second = res[second_begin:second_end]

                print(hour + ":" + minite + ":" + second)
                cur_time = hour + ':' + minite;
                codey.display.show(cur_time)
        except:
            print("get error data")
    else:
        codey.led.show(0,0,0)
```

mqtt - Transmisie coadă de mesaje prin telemetrie

Tabelul 25 conține clasa *mqtt.MQTTClient* și funcțiile de utilizare a acesteia.

Tabelul 25: Funcțiile *Python* pentru *mqtt*

Funcția	Acțiune
class mqtt.MQTTClient (client_id, server, port=0, user=None, password=None, keepalive=0, ssl=False, ssl_params={}) -clasă	Instanțarea obiectului de interfață al clientului MQTT, parametrii: • *client_id* – reprezintă șirul unic de ID client utilizat la conectarea la broker. Dacă *client_id* are o lungime zero sau None, atunci unul va fi generat aleatoriu. În acest caz, parametrul *clean_session* al funcției de conectare trebuie să fie *True*. • *server* -reprezintă numele gazdei sau adresa IP a serverului la distanță. • *port* (opțional) - reprezintă portul de rețea al gazdei serverului la care vă conectați. Numărul de port implicit este 1883. Vă rugăm să rețineți că numărul de port implicit al MQTT pe SSL / TLS este 8833. • *user* (opțional) - reprezintă numele de utilizator înregistrat pe server. • *password* (opțional) – reprezintă parola înregistrată pe server. • *keepalive* (opțional) - reprezintă valoarea de expirare a timpului de păstrare a clientului. Valoarea implicită este de 60 s. • *ssl* (opțional) - dacă activați suportul SSL / TLS. • ssl_*params* (opțional) - reprezintă parametrul SSL / TLS.
connect(clean_session=True)	Conectează clientul la server, aceasta este o funcție de blocare, parametrii: *clean_session* -reprezintă o valoare booleană care determină tipul de client. Dacă ia valoarea *True*, brokerul va elimina toate informațiile despre acest client atunci când se deconectează. Dacă este *False*,

	clientul este un client durabil, iar informațiile despre abonament și mesajele din coadă vor fi păstrate atunci când clientul se deconectează.
reconnect()	Reconectare la server folosind detaliile furnizate anterior. Trebuie să apelăm conectarea înainte de a apela această funcție.
disconnect()	Deconectare de la server.
ping()	Testează conectivitatea dintre server și client.
set_last_will(topic, msg, retain=False, qos=0)	Setează voința de a fi trimisă la server. Dacă clientul se deconectează fără a apela *disconnect (),* serverul va posta un mesaj în numele său, parametrii: • *topic* – reprezintă subiectul postării. • *msg* - va trimite mesajul. • *retain* - dacă este setat la *True*, mesajul de testare va fi setat la ultimul mesaj bun / rezervat cunoscut pentru subiect. • *qos* - este utilizat pentru calitatea serviciului.
publish(topic, msg, retain=False, qos=0)	Un mesaj este trimis de la client către agent și apoi trimis de la agent către orice client care se abonează la subiectul corespunzător, parametrii: *topic* – reprezintă subiectul mesajului ce ar trebui să fie postat. *msg* - mesajul real de trimis. *Retain* - dacă este setat la *True*, mesajul de testare va fi setat la ultimul mesaj bun / rezervat cunoscut pentru subiect. *qos* - nivelul de calitate al serviciului utilizat.
subscribe(topic, qos=0)	Acest modul oferă câteva funcții de asistență pentru abonare și procesare directă a mesajelor. De exemplu *set_callback*, parametrii: • *topic* - reprezintă subiectul mesajului de abonat. • *qos* - nivelul de calitate al serviciului de utilizat.
set_callback(f)	Setează funcția de apel invers pentru subiectul abonament, care este apelat atunci

	când serverul răspunde la solicitarea noastră de abonament, parametrul: *f* – reprezintă funcția de apel invers.
wait_msg()	Așteaptă până când serverul nu are mesaje în așteptare. Această funcție este o funcție de blocare.
check_msg()	Verifică dacă serverul are mesaje în așteptare. Dacă nu, revine direct, dacă există, efectuează aceeași procesare ca și funcția *wait_msg*.

Exemplu:

```
from mqtt import MQTTClient
import codey
import time

MQTTHOST = "mq.makeblock.com"
MQTTPORT = 1883

# Completați după cum doriți
client_id = "20180911203800"

# Exemplu de cale
Topic = "/sensors/temperature/#"

mqttClient = MQTTClient(client_id, MQTTHOST, port=MQTTPORT,
user='test', password='test', keepalive=0, ssl=False)

# Conectarea la serverul MQTT
def on_mqtt_connect():
    mqttClient.connect()

# publicarea mesajului
def on_publish(topic, payload, retain=False, qos = 0):
    mqttClient.publish(topic, payload, retain, qos)

# funcția de procesare a mesajului
def on_message_come(topic, msg):
    print(topic + " " + ":" + str(msg))
    codey.display.show(msg)

# mesaj de abonare
def on_subscribe():
    mqttClient.set_callback(on_message_come)
    mqttClient.subscribe(Topic, qos = 1)
```

```
# Completați aici ssid-ul și parola routerului.
codey.wifi.start('wifi_ssid', 'password')
codey.led.show(0,0,0)
codey.display.show(0)
while True:
    if codey.wifi.is_connected():
        on_mqtt_connect()
        on_subscribe()
        codey.led.show(0,0,255)
        while True:
            # Blocarea așteptării mesajului
            on_publish("/sensors/temperature/home",
str(38), qos = 1)
            mqttClient.wait_msg()
            time.sleep(1)
    else:
        codey.led.show(0,0,0)
```

API Python pentru extensia modulelor Neuron

dc_motor_driver - Driver dual motor DC

Tabelul 26 conține funcția de utilizare a *dc_motor_driver*.

Tabelul 26: Funcția *Python* pentru *dc_motor_driver*

Funcția	Acțiune
dc_motor_driver.set_power(speed, ch = 0)	Setează puterea pentru driverul motorului în fiecare canal, parametrii: • *speed* - se referă la valoarea de putere a motorului controlat, intervalul de parametri este -100 ~ 100. • *ch* - se referă la numărul de canal al motorului controlat, intervalul de parametri este 0 ~ 2 și 0: reprezintă ambele sloturi , 1: pentru slotul 1, 2: pentru slotul 2.

Exemplu:

```
import codey
import neurons
import event

@event.button_a_pressed
def on_button_a_pressed():
    print("button a event succeeded")
    neurons.dc_motor_driver.set_power(100, 1)

@event.button_b_pressed
def on_button_b_pressed():
    print("button b event succeeded")
    neurons.dc_motor_driver.set_power(100, 2)

@event.button_c_pressed
def on_button_c_pressed():
    print("button c event succeeded")
    neurons.dc_motor_driver.set_power(100, 0)
    neurons.dc_motor_driver.set_power(100, 1, 2)
```

servo_driver - Dual Servo Driver

Tabelul 27 conține funcția de utilizare a *servo_driver*.

Tabelul 27: Funcția *Python* pentru *servo_driver*

Funcția	Acțiune
servo_driver.set_angle(position, ch = 0)	Setează puterea pentru servo-driver în fiecare canal, parametrii: • *position* - se referă la valoarea unghiului de rotație a servo-driver – ului controlat, intervalul de parametri este 0 ~ 180. • *ch* - se referă la numărul de canal controlat, intervalul de parametri este 0 ~ 2 și 0: reprezintă ambele sloturi, 1: pentru canalul 1, 2: pentru al 2-lea canal.

Exemplu:

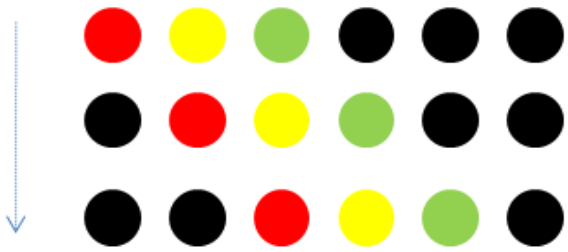
```
import codey
import neurons
import event
import time

neurons.servo_driver.set_angle(0, 0)
time.sleep(1)

@event.button_a_pressed
def on_button_a_pressed():
    print("button a event succeeded")
    neurons.servo_driver.set_angle(100, 1)

@event.button_b_pressed
def on_button_b_pressed():
    print("button b event succeeded")
    neurons.servo_driver.set_angle(100, 2)

@event.button_c_pressed
def on_button_c_pressed():
    print("button c event succeeded")
    neurons.servo_driver.set_angle(100, 0)
```

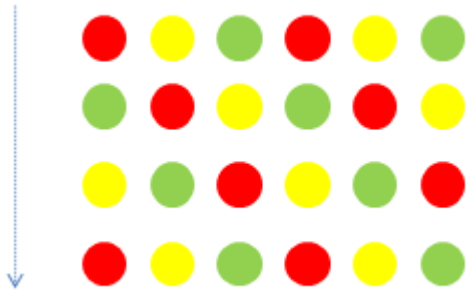
led_strip - Driver de bandă LED

Tabelul 28 conține funcțiile *Python* destinate lucrului cu driverul de bandă LED.

Tabelul 28: Funcțiile *Python* pentru *driverul de bandă LED*

Funcția	Acțiune
led_strip.set_single(index, red_value, green_value, blue_value)	Setează culoarea fiecărei lumini de pe banda LED, parametrii: • *index* - setează ordinea luminii Nu, intervalul de parametri este 1 ~ valoarea luminilor totale de pe banda LED. • *red_value* - setează valoarea roșu a LED-ului, intervalul de parametri este 0 ~ 255, 0 înseamnă că nu există culoare roșie, 255 înseamnă cea mai strălucitoare culoare roșie. • *green_value* - setează valoarea verde a LED-ului, intervalul de parametri este 0 ~ 255, 0 înseamnă că nu există culoare verde, 255 înseamnă cea mai strălucitoare culoare verde. • *blue_value* - setează valoarea LED albastru, intervalul de parametri este 0 ~ 255, 0 înseamnă că nu există culoare albastră, 255 înseamnă cea mai strălucitoare culoare albastră.
led_strip.set_all(red_value, green_value, blue_value)	Setează culoarea pentru toate luminile. Parametrii: • *red_value* - setarea valorii roșu a LED-ului, intervalul de parametri este 0 ~ 255, 0 înseamnă că nu există culoare roșie, 255 înseamnă cea mai strălucitoare culoare roșie. • *green_value* - setarea valorii verde a LED-ului, intervalul de parametri este 0 ~ 255, 0 înseamnă că nu există

	culoare verde, 255 înseamnă cea mai strălucitoare culoare verde. • *blue_value* - setarea valorii albastru a LED- ului, intervalul parametrilor este 0 ~ 255, 0 înseamnă că nu există culoare albastră, 255 înseamnă cea mai strălucitoare culoare albastră.
led_strip.set_effect(mode, speed, list)	Setează efectul benzii LED. Parametrii: • *mode* - efect mod, intervalul parametrilor este 0 ~ 5. *0:* - înseamnă modul static: starea luminilor păstrează ultima setare. *1:* - înseamnă modul de rulare: luminile N din față se aprind mai întâi ca culoare de setare, apoi luminile N se deplasează la 2 ~ N + 1 și prima se stinge, apoi 3 ~ N + 2 și primele două lumini se sting, ca în imaginea de mai jos:  2: - înseamnă modul de repetare: luminile N din față se aprind mai întâi ca culoare de setare, iar luminile de repaus vor copia starea respectivă până la ultima lumină, la fel ca în imaginea de mai jos:  3: - înseamnă modul marcaj: lumina N se aprinde și apoi se deplasează în mod repetat la o viteză de setare, după cum se arată în imaginea de mai jos:



4: - înseamnă modul de respirație: luminile se schimbă la viteza respirației umane, adică se aprind / opresc la fiecare trei secunde.
5: - înseamnă modul gradient: toate luminile de pe bandă își schimbă culoarea treptat în noua culoare de setare într-un anumit timp de setare.

- *speed* - schimbarea dinamică a vitezei, intervalul parametrilor este de 0 ~ 8, 0 - înseamnă viteza cea mai mică și 8 - este cea mai rapidă (funcționează numai atunci când există setarea schimbării dinamice a stării luminilor).
- *list* - lista parametrilor modificabili, intervalul parametrilor este 0 ~ 8 , primul parametru înseamnă prima culoare de lumină, al doilea parametru înseamnă a doua culoare de lumină și așa mai departe; Și parametrii de culoare sunt după cum urmează: *black(0x00), red(0x01), orange(0x02), yellow(0x03), green(0x04), cray(0x05), blue(0x06), purple(0x07) and while(0x08).*

Exemplu:

```
import codey
import neurons
import event
import time

neurons.led_strip.set_all(0, 0, 255)
time.sleep(1)
```

```
@event.button_a_pressed
def on_button_a_pressed():
    print("button a event successed")
    neurons.led_strip.set_all(0, 0, 0)
    neurons.led_strip.set_single(1, 255, 0, 0)
    time.sleep(1)
    neurons.led_strip.set_all(0, 0, 0)
    neurons.led_strip.set_single(2, 255, 0, 0)
    time.sleep(1)
    neurons.led_strip.set_all(0, 0, 0)
    neurons.led_strip.set_single(3, 255, 0, 0)
    time.sleep(1)

@event.button_b_pressed
def on_button_b_pressed():
    print("button b event successed")
    neurons.led_strip.set_effect(0, 8, (1,6,8,1,6,8,1,6,8))
    time.sleep(3)
    neurons.led_strip.set_effect(1, 8, (1,6,8,1,6,8,1,6,8))
    time.sleep(3)
    neurons.led_strip.set_effect(2, 8, (1,6,8,1,6,8,1,6,8))
    time.sleep(3)
    neurons.led_strip.set_effect(3, 8, (1,6,8,1,6,8,1,6,8))
    time.sleep(3)
    neurons.led_strip.set_effect(4, 8, (1,6,8,1,6,8,1,6,8))
    time.sleep(3)
    neurons.led_strip.set_effect(5, 8, (1,6,8,1,6,8,1,6,8))
    time.sleep(3)
@event.button_c_pressed
def on_button_c_pressed():
    print("button c event successed")
    neurons.led_strip.set_effect(0, 5, (1,1,1,1,1,1,1,1,1))
    time.sleep(3)
    neurons.led_strip.set_effect(1, 5, (1,1,1,1,1,1,1,1,1))
    time.sleep(3)
    neurons.led_strip.set_effect(2, 5, (1,1,1,1,1,1,1,1,1))
    time.sleep(3)
    neurons.led_strip.set_effect(3, 5, (1,1,1,1,1,1,1,1,1))
    time.sleep(3)
    neurons.led_strip.set_effect(4, 5, (1,1,1,1,1,1,1,1,1))
    time.sleep(3)
    neurons.led_strip.set_effect(5, 5, (1,1,1,1,1,1,1,1,1))
    time.sleep(3)
```

led_panel - panoul LED RGB

Tabelul 29 conține funcțiile *Python* destinate lucrului cu panoul LED RGB.

Tabelul 29: Funcțiile *Python* pentru panoul LED RGB

Funcția	Acțiune
led_panel.set_all (red_value, green_value, blue_value)	Setează și afișează culoarea tuturor luminilor de pe panou. Parametrii: • *red_value* - setează valoarea roșu a LED-ului, intervalul de parametri este 0 ~ 255, 0 înseamnă că nu există culoare roșie, 255 înseamnă cea mai strălucitoare culoare roșie. • *green_value* - setează valoarea verde a LED-ului, intervalul de parametri este 0 ~ 255, 0 înseamnă că nu există culoare verde, 255 înseamnă cea mai strălucitoare culoare verde. • *blue_value* - setează valoarea albastră a LED-ului, intervalul de parametri este 0 ~ 255, 0 înseamnă că nu există culoare albastră, 255 înseamnă cea mai strălucitoare culoare albastră.
led_panel.set_pixel (x, y, red_value, green_value, blue_value)	Setează culoarea pentru fiecare pixel de pe panou. Parametrii: • *x* - poziția pixelului x pe panou, intervalul de parametri este de la 0 la 7. • *y* - poziția pixelului y pe panou, intervalul de parametri este de la 0 la 7. • *red_value* - setarea valorii roșu a LED-ului, intervalul de parametri este 0 ~ 255, 0 înseamnă că nu există culoare roșie, 255 înseamnă cea mai strălucitoare culoare roșie. • *green_value* - setarea valorii verde a LED-ului, intervalul de parametri este 0 ~ 255, 0 înseamnă că nu există culoare verde, 255 înseamnă cea mai strălucitoare culoare verde.

	• *blue_value* - setarea valorii albastru a LED-ului, intervalul de parametri este 0 ~ 255, 0 înseamnă că nu există culoare albastră, 255 înseamnă cea mai strălucitoare culoare albastră.
led_panel.show_image (list, mode = 0)	Setează conținutului afișajului ca mod parametru imagine. Parametrii: • *list* - lista parametrilor modificabili, intervalul parametrilor este 0 ~ 8 , primul parametru înseamnă prima culoare de lumină, al doilea parametru înseamnă a doua culoare de lumină și așa mai departe; iar parametrii de culoare sunt după cum urmează: black(0x00), red(0x01), orange(0x02), yellow(0x03), green(0x04), cray(0x05), blue(0x06), purple(0x07) și while(0x08). • *mode* - modul de afișare a conținutului modului, intervalul parametrilor este 0 ~ 3 0: înseamnă modul emergent, setarea imaginii va fi afișată direct. 1: înseamnă modul de ștergere, imaginea originală dispare treptat și imaginea de setare nouă se va afișa treptat și vertical. 2: înseamnă modul în mișcare la stânga, imaginea originală se deplasează la stânga și dispare treptat, iar imaginea de setare nouă se va deplasa la stânga până când se afișează întreaga imagine. 3: înseamnă modul în mișcare dreapta, imaginea originală se deplasează spre dreapta și dispare treptat, iar imaginea de setare nouă se va deplasa spre dreapta până când se afișează întreaga imagine.
led_panel.set_animation(frame_index, list)	Setarea conținutului animației pe panou. Parametrii: • *frame_index* – reprezintă indexul cadrului de animație, intervalul de parametri este 0 ~ 3; 0 - înseamnă primul cadru, 1 înseamnă al doilea și așa mai departe.

	• *list* - reprezintă lista parametrilor modificabili, intervalul parametrilor este 0 ~ 8 , primul parametru înseamnă prima culoare de lumină, al doilea parametru înseamnă a doua culoare de lumină și așa mai departe; Și parametrii de culoare sunt după cum urmează: *black(0x00), red(0x01), orange(0x02), yellow(0x03), green(0x04), cray(0x05), blue(0x06), purple(0x07) și while(0x08).*
led_panel.show_animation(frame_speed, mode)	Afișează setarea cadrului de animație prin *set_animation*. Parametrii: • *frame_speed* – reprezintă viteza de comutare a conținutului cadrului de animație, intervalul de parametri este 0 ~ 2 0: - înseamnă viteză lentă pe care cadrul de animație îl rulează la fiecare secundă 1: - înseamnă viteza normală pe care cadrul de animație o rulează la fiecare jumătate de secundă 2: - înseamnă viteză rapidă pe care cadrul de animație o rulează la fiecare 0,2 secunde. • *mode* - modul de schimbare a cadrului de mod, intervalul parametrilor este 0 ~ 3 0: - înseamnă modul emergent, setarea imaginii va fi afișată direct. 1: - înseamnă modul de ștergere, imaginea originală dispare treptat și imaginea de setare nouă se va afișa treptat și vertical. 2: - înseamnă modul în mișcare la stânga, imaginea originală se deplasează la stânga și dispare treptat, iar imaginea de setare nouă se va deplasa la stânga până când se afișează întreaga imagine. 3: - înseamnă modul în mișcare spre dreapta, imaginea originală se deplasează spre dreapta și dispare treptat, iar imaginea de setare nouă se va deplasa spre dreapta până când se afișează întreaga imagine.
led_panel.show_string(red_value, green_value, blue_value, list)	Afișează șirul ca culoare de setare. Parametrii:

	• *red_value* - setează valoarea roșu a LED-ului, intervalul de parametri este 0 ~ 255, 0 înseamnă că nu există culoare roșie, 255 înseamnă cea mai strălucitoare culoare roșie. • *green_value* - setează valoarea verde a LED-ului, intervalul de parametri este 0 ~ 255, 0 înseamnă că nu există culoare verde, 255 înseamnă cea mai strălucitoare culoare verde. • blue_value - setează valoarea albastru a LED-ului, intervalul parametrilor este 0 ~ 255, 0 înseamnă că nu există culoare albastră, 255 înseamnă cea mai strălucitoare culoare albastră. • *list* - lista parametrilor modificabili, primul caracter, al doilea caracter, al treilea caracter ...
led_panel.clear()	Curățarea (ștergerea) panoului frontal Codey.

Exemplu:

```
import codey
import neurons
import event
import time

neurons.led_panel.clear()
neurons.led_panel.set_all(0, 0, 255)
time.sleep(1)
neurons.led_panel.clear()

@event.button_a_pressed
def on_button_a_pressed():
    print("Evenimentul butonului A reușit ")
    neurons.led_panel.set_pixel(0, 0, 255, 0, 0)
    time.sleep(1)
    neurons.led_panel.set_pixel(4, 4, 255, 0, 0)
    time.sleep(1)
    neurons.led_panel.set_pixel(7, 7, 255, 0, 0)
    time.sleep(1)
    neurons.led_panel.set_pixel(0, 6, 255, 0, 0)
    time.sleep(1)

@event.button_b_pressed
```

```
def on_button_b_pressed():
    print("Evenimentul butonului B reușit ")
    neurons.led_panel.show_image([1,6,8,0,0,0,1,6,8],0)
    time.sleep(1)
    neurons.led_panel.show_image([1,1,1,1,1,1,1,1,1],1)
    time.sleep(1)
    neurons.led_panel.show_image([6,6,6,6,6,6,6,6,6],2)
    time.sleep(1)
    neurons.led_panel.show_image([8,8,8,8,8,8,8,8,8],3)
    time.sleep(1)

@event.button_c_pressed
def on_button_c_pressed():
    print("Evenimentul butonului C reușit")
    neurons.led_panel.set_animation(0, (1,6,8,1,6,8,0,0,0))
    neurons.led_panel.set_animation(1, (6,6,6,6,6,6,6,6,6))
    neurons.led_panel.set_animation(2, [6,6,6,6,6,6,6,6,6])
    neurons.led_panel.set_animation(3, (8,8,8,8,8,8,8,8,8))
    neurons.led_panel.show_animation(1, 2)
    time.sleep(6)
    neurons.led_panel.show_string(255, 0, 0, "Salut!")
    time.sleep(4)
    neurons.led_panel.show_string(255, 0, 0, (100))
    time.sleep(4)
    neurons.led_panel.show_string(255, 0, 0, (1,2,3))
```

buzzer – Sirenă

Tabelul 30 conține funcții ce pot fi utilizate în programarea sirenei.

Tabelul 30: Funcțiile *Python* pentru *buzzer*

Funcția	Acțiune
buzzer.play_note(note_num, beat = None)	Redă notele muzicale definite digital. Parametrii: • *note_num* - valoare numerică, interval de valori 48 - 72 sau tip șir, cum ar fi C4. • *beat* - indică numărul de bătăi, valoarea implicită se joacă întotdeauna. notele și frecvența sunt după cum urmează:

	['C2','65'], ['D2','73'], ['E2','82'], ['F2','87'], ['G2','98'], ['A2','110'], ['B2','123'], ['C3','131'], ['D3','147'], ['E3','165'], ['F3','175'], ['G3','196'], ['A3','220'], ['B3','247'], ['C4','262'], ['D4','294'], ['E4','330'], ['F4','349'], ['G4','392'], ['A4','440'], ['B4','494'], ['C5','523'], ['D5','587'], ['E5','659'], ['F5','698'], ['G5','784'], ['A5','880'], ['B5','988'], ['C6','1047'], ['D6','1175'], ['E6','1319'], ['F6','1397'], ['G6','1568'], ['A6','1760'], ['B6','1976'], ['C7','2093'], ['D7','2349'], ['E7','2637'], ['F7','2794'], ['G7','3136'], ['A7','3520'], ['B7','3951'], ['C8','4186'], ['D8','4699'],
buzzer.play_tone(frequency, time = None)	Redă tonul frecvenței de setare. Parametrii: • *frequency* - date numerice, ce indică frecvența sunetului care este redat și domeniul său de valori este de 0 ~ 5000. • *time* - date numerice, care indică timpul de redare (în milisecunde - ms) și intervalul său de valori este 0 ~ limita intervalului de valori.
buzzer.rest(number)	Oprește ritmul. Parametrul: • *number* - date numerice, numărul de bătăi întrerupte, intervalul său de valori este 0 ~ limita intervalului de valori.
buzzer.tempo - constantă	Date numerice, care indică natura vitezei de redare, în bmp (bătăi pe minut), care este lungimea fiecărei bătăi. Gama sa de valori este de 6 ~ 600. Valoarea implicită este 60, ceea ce înseamnă că durata unei bătăi este 1 secunda. Bătăile funcțiilor *rest* și *play_note* sunt afectate de această constantă.

Exemplu:

```
import codey
import time
import neurons

codey.display.show("hello")

neurons.buzzer.play_note(48, 1)
neurons.buzzer.rest(1)
codey.display.show("note")
codey.display.clear()
```

```
neurons.buzzer.play_note("C4", 1)
neurons.buzzer.rest(1)
codey.display.show("C4")
codey.display.clear()
neurons.buzzer.play_tone(1000, 2)
neurons.buzzer.rest(1)
codey.display.show("tone")
codey.display.clear()

while True:
    neurons.buzzer.tempo = 60
    print("tempo:", end = "")
    print(neurons.buzzer.tempo)
    neurons.buzzer.play_note("C4", 1)
    neurons.buzzer.rest(2)
    neurons.buzzer.tempo = 240
    neurons.buzzer.play_note("C4", 1)
    neurons.buzzer.rest(2)
```

button – Buton

Tabelul 31 conține funcția *Python* destinată programării butoanelor.

Tabelul 31: Funcția *Python* pentru programarea butoanelor

Funcția	Acțiune
button.is_pressed()	Obținem starea curentă a butonului; rezultatul va fi *True*: dacă butonul apăsat sau *False*: dacă butonul nu este apăsat.

Exemplu:
 	
`import neurons`
 	
`while True:`
`  if neurons.button.is_pressed():`
`  print("pressed!")`	

funny_touch – Atingere amuzantă

Dispozitivul *Funny Touch* (fig. 7) poate fi conectat la orice obiect conductiv (cum ar fi bananele și apa) și-l transformă într-un comutator tactil. Un efect interactiv simplu și interesant poate fi obținut prin detectarea stării de conducere între comutatoarele amuzante și firul GND.

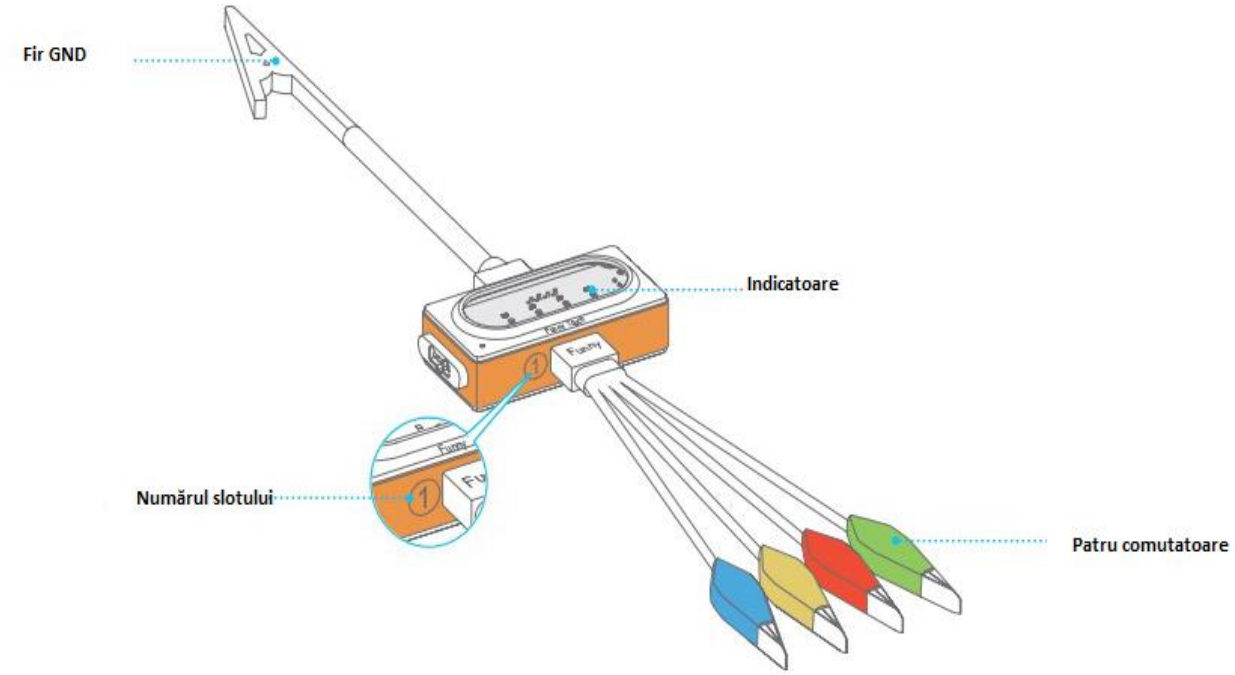


Figura 7: Dispozitivul *Funny Touch*

Mai jos sunt indicați pașii de utilizare a dispozitivului *Funny Touch:*

- Conectăm comutatorul amuzant la slotul 1 și firul GND la slotul 2.
- Clipăm un comutator amuzant pe un obiect conductiv.
- Ținem clema metalică a firului GND și atingem obiectul conductor cu cealaltă mână, indicatorul relevant se va aprinde și blocul va trimite un semnal de pornire.

Notă: Clema aligatorului este ascuțită, nu vă decupați cu comutatorul amuzant sau cu clema firului GND, vă poate face rău.

Tabelul 32 conține funcțiile *Python* ce permit programarea utilizând un astfel de dispozitiv.

Tabelul 32: Funcțiile *Python* pentru *Funny Touch*

Funcția	Acțiune
funny_touch.is_red_touched()	Indiferent dacă comutatorul roșu este atins sau nu, rezultatul va fi *True*: da, este atins sau *False*: nu, nu este atins.
funny_touch.is_green_touched()	Indiferent dacă comutatorul verde este atins sau nu, rezultatul va fi *True*: da, este atins sau *False*: nu, nu este atins.
funny_touch.is_yellow_touched()	Indiferent dacă comutatorul galben este atins sau nu, rezultatul va fi *True*: da, este atins sau *False*: nu, nu este atins.
funny_touch.is_blue_touched()	Indiferent dacă comutatorul albastru este atins sau nu, rezultatul va fi *True*: da, este atins sau *False*: nu, nu este atins.

Exemplu:

```
import codey
import time
import event
import neurons

@event.start
def start_cb():
    while True:
        if neurons.funny_touch.is_blue_touched():
            print("blue touched")
        if neurons.funny_touch.is_red_touched():
            print("red touched")
        if neurons.funny_touch.is_green_touched():
            print("green touched")
        if neurons.funny_touch.is_yellow_touched():
            print("yellow touched")

        time.sleep(0.1)
```

ultrasonic_sensor - Senzor ultrasunet

Tabelul 33 conține funcția destinată lucrului cu senzorul ultrasunet.

Tabelul 33: Funcția *Python* pentru *ultrasonic_sensor*

Funcția	Acțiune
ultrasonic_sensor.get_distance()	Obținem distanța (cm) între obstacolul din față și senzorul cu ultrasunete; rezultatul este în virgulă mobilă, variind de la 0 ~ 300 cm; dar măsurăm intervalele de distanță de 3 ~ 300 cm, deoarece detectarea nu este suficient de exactă în termen de 3 cm.

Exemplu:
 	
`import codey`
`import time`
`import event`
`import neurons`
 	
`@event.start`
`def start_cb():`
`  while True:`
`  print(neurons.ultrasonic_sensor.get_distance())`
`  time.sleep(0.2)`	

gyro_sensor - Ghid de utilizare a senzorului giroscopic

Imaginea de mai jos (fig. 8) reprezintă sistemul de coordonate al modulului Gyro.

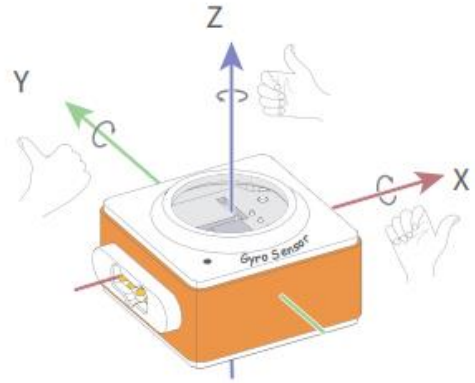


Figura 8: Sistemul de coordonate al modulului *Gyro*.

Tabelul 34 conține funcțiile *Python* ce permit utilizarea modulului *Gyro*.

Tabelul 34: Funcțiile *Python* pentru *gyro_sensor*

Funcția	Acțiune
gyro_sensor.get_roll()	Obținem unghiul *Euler*, intervalul de date returnat este de -90 ~ 90.
gyro_sensor.get_pitch()	Obținem pasul unghiului *Euler*, intervalul de date returnat este -180 ~ 180.
gyro_sensor.get_pitch()	Obținem valoarea unghiului *Euler*, intervalul de date returnat este -32768 ~ 32767, deoarece senzorul giroscopic este un senzor cu șase axe, nu există busolă electronică. Deci, de fapt, unghiul de fală este doar integralul vitezei unghiulare a axei Z. Dacă dorim să obținem un unghi adevărat de falcă, acest API nu este potrivit pentru utilizare.

gyro_sensor.is_shaked()	Verifică dacă senzorul giroscopic este agitat, valoarea returnată este o valoare booleană, unde *True* înseamnă că senzorul giroscopic este agitat, iar *False* înseamnă că senzorul giroscopic nu este agitat.
gyro_sensor.get_acceleration(axis)	Obținem valorile de accelerație ale celor trei axe în g, parametrul: • *axis* - șirul de axe, notate cu x, y, z reprezentând axa definită de senzorul giroscopic
gyro_sensor.get_gyroscope(axis)	Obținem valorile vitezei unghiulare ale celor trei axe în ° / sec, parametrul: • *axis* – șirul de axe notate cu x, y, z reprezentând axa definită de senzorul giroscopic.

Exemplul 1:

```
import rocky
import event
import neurons

@event.button_a_pressed
def on_button_a_callback():
    codey.stop_other_scripts()
    codey.display.show("pit")
    while True:
        print(neurons.gyro_sensor.get_pitch())
        time.sleep(0.05)

@event.button_b_pressed
def on_button_b_callback():
    codey.stop_other_scripts()
    codey.display.show("rol")
    while True:
        print(neurons.gyro_sensor.get_roll())
        time.sleep(0.05)

@event.button_c_pressed
def on_button_c_callback():
    codey.stop_other_scripts()
    codey.display.show("yaw")
    while True:
        print(neurons.gyro_sensor.get_yaw())
        time.sleep(0.05)
```

Exemplul 2:

```
import rocky
import event
import neurons

@event.start
def start_cb():
    codey.display.show("sha")
    while True:
        print(neurons.gyro_sensor.is_shaked())
        time.sleep(0.2)
```

Exemplul 3:

```
import rocky
import event
import neurons

@event.start
def start_cb():
    while True:
        print("gyro z:", end = "")
        print(neurons.gyro_sensor.get_gyroscope("z"))
        print("accel z:", end = "")
        print(neurons.gyro_sensor.get_acceleration("z"))
        time.sleep(0.2)
```

pir_sensor - Senzor PIR

Tabelul 35 include funcția de activare a senzorului PIR.

Tabelul 35: Funcția Python pentru *pir_sensor*

Funcția	Acțiune
pir_sensor.is_activated()	Obținem rezultatul detectării de la senzor. Rezultatul va fi *True*: detectează omul din apropiere sau *False*: nu detectează omul din apropiere.
Exemplu: `import codey`	

```
import time
import event
import neurons

@event.start
def start_cb():
    while True:
        print(neurons.pir_sensor.is_activated())
        time.sleep(0.2)
```

soil_moisture - Umiditatea solului

Tabelul 36 conține funcția *Python* ce permite detectarea umidității solului.

Tabelul 36: Funcția *Python* pentru *soil_moisture*

Funcția	Acțiune
soil_moisture.get_value()	Obținem umiditatea solului detectată, variind de la 0 la 100; cu cât valoarea este mai mare, cu atât este mai mare umiditatea.

Exemplu:

```
import codey
import time
import event
import neurons

@event.start
def start_cb():
    while True:
        print(neurons.soil_moisture.get_value())
        time.sleep(0.2)
```

Concluzii și recomandări

Acest ghid de utilizare a fost creat în cadrul Laboratorului de Cercetare „*Intelegenţa Artificială Creativă*” a Universităţii Pedagogice de Stat „Ion Creangă” cu scopul de a ajuta şi încuraja profesorii și elevii să studieze disciplina opțională *Robotica*. Robotica creează un context educațional plăcut și atrăgător care promovează educația STEM și îmbunătățește procesul de învățare [5, 6]. De asemenea, *Robotica Educaţională* creează un mediu de învățare în care elevii pot găsi și dezvolta soluții pentru a lucra cu probleme reale datorită prezenței senzorilor și a dispozitivelor de acționare [7], poate fi un instrument care ajută elevii și profesorii să facă învățarea mai activă și elevii mai motivați [8]. În plus, Robotica Educaţională este considerată o modalitate care poate fi utilizată pentru a construi abilitățile necesare succesului în secolul 21 [9].

Este cunoscut faptul că multe instituții de învățământ general nu dispun de echipamente și dispozitive moderne că să permită profesorilor să realizeze această disciplină opțională. O altă cauză de ce nu se studiază această disciplină este și lipsa materialelor didactice necesare. Pentru elucidarea acestor cauze propunem procurarea doar a unui roboțel Codey Rocky care costă în jur de 150 de euro care să fie utilizat pentru integrarea disciplinelor în procesul educaţional și pentru a susține tehnologic realizarea proiectelor de tip STEAM care au apărut în curricula diverselor discipline școlare. De asemenea propunem utilizarea acestui ghid pentru predarea disciplinei opționale menționate mai sus. Este destinat publicului larg, de la elevii din gimnaziu, elevii din liceu, studenți, adoslescenți, profesori, părinți etc.

Bibliografie

1. Despre mBlock5 disponibil online: https://www.mblock.cc/doc/en/part-one-basics/basics.html
2. mBlock3 vs mBlock disponibil online: https://www.mblock.cc/doc/en/mblock3/mblock3-vs-mblock5.html
3. Comunitatea Makeblock: http://planet.mblock.cc/gallery/2549
4. Caietul elevului disponibil: https://www.generationrobots.com/media/makeblock/codey-rocky/StudentsBook.pdf
5. KHANLARI, A.; KIAIE, F.M. Using Robotics for STEM Education in Primary/Elementary Schools: Teachers' Perceptions. In: 10th International Conference on Computer Science and Education, ICCSE. 2015.
6. KHANLARI, A. Effects of Robotics on 21st Century Skills. In: European Scientific Journal, 2013. 9: p. 26-36.
7. Vascan, T. Promovarea educaţiei STEAM prin intermediul Roboticii Educaţionale, Conferința științifică internațională "Abordări inter/transdisciplinare în predarea științelor reale (concept STEAM)", UST, Facultatea de Fizică, Matematică și Tehnologii Informaționale, Departamentul Didactica Științelor, 29-30 octombrie 2021, volumul I, pp. 330-335.
8. Vascan, T. Robotica în educația STEAM În: Acta et Commentationes. Științe ale Educației. Revistă științifică Nr.2(28) (2022). Chișinău: Universitatea de Stat din Tiraspol, 2022. p.41-49. ISSN 1857-0623. (Categoria B).
9. KHANLARI, A. Effects of Robotics on 21st Century Skills. In: European Scientific Journal, 2013. 9: p. 26-36.
10. MIDI note numbers and center frequencies: https://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies
11. Definițiile notelor muzicale: https://en.scratch-wiki.info/wiki/Play_Note_()_for_()_Beats_(block)