

Systemy wbudowane dla automatyki Lab06

Python: JSON, TCP, HTTP

Zostawiłem treść W10, żeby mieć wszystko w jednym pliku.

Dane sieciowe w trakcie laborek:

Adres pi4.local lub 10.0.0.198, port **80** (HTTP) oraz **8052** (TCP)

WiFi: SSID=SWA, Klucz=systemywbudowane

Dodatkowo pojawił się port **8053** serwera TCP, który pozwala na komunikację między klientami.

Dane sieciowe do wykładu zdalnego i zabaw poza labem:

Adres: agrosensor.jumpingcrab.com

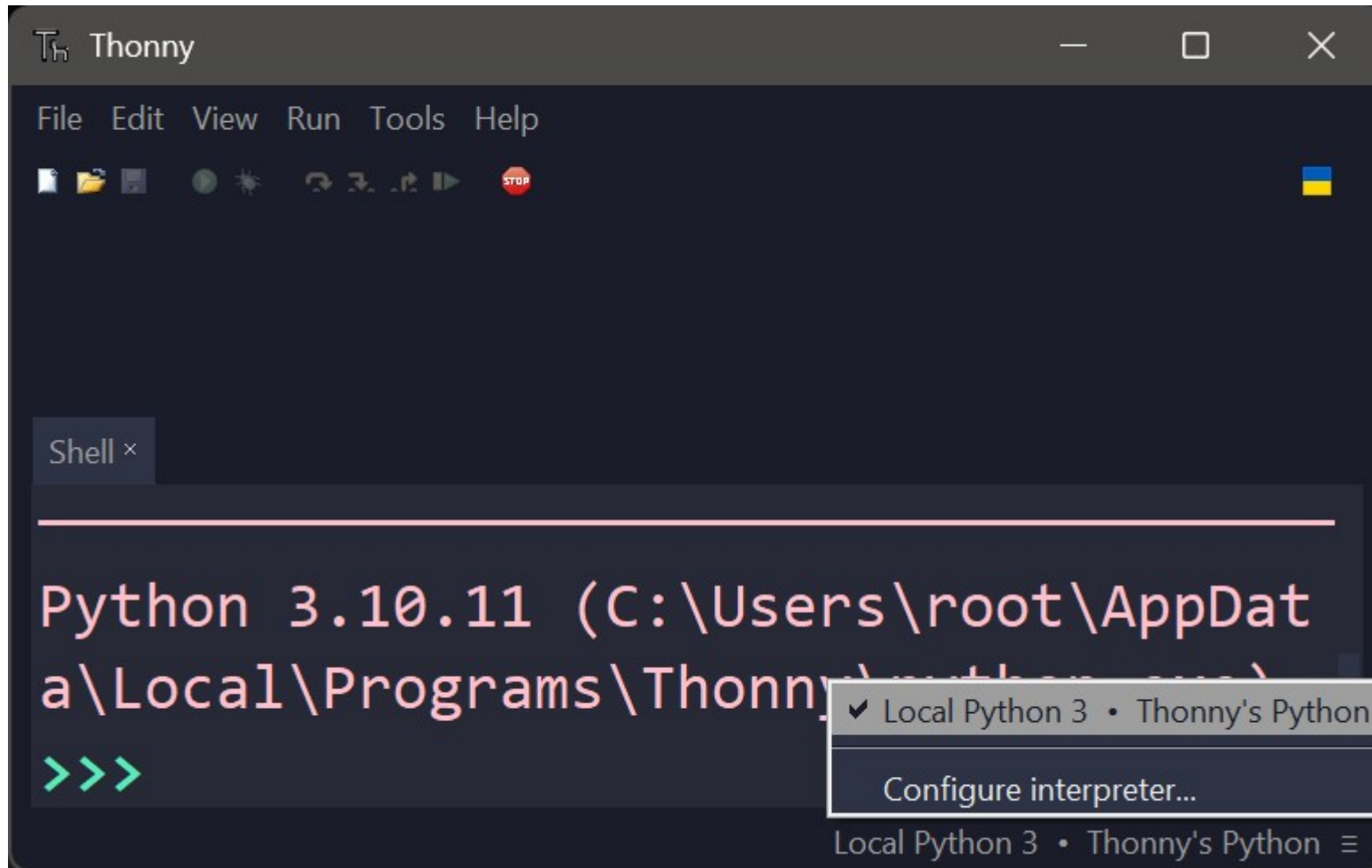
Port 8051 (HTTP) oraz 8052 (TCP)

W przeglądarce wpisz adres: <http://agrosensor.jumpingcrab.com:8051/>

i sprawdź, czy się wyświetla strona testowa.

Dodatkowo port 8053 dla serwera TCP, który pozwala na komunikację między klientami.

Oprogramowanie: Thonny (Windows, Linux), lokalny interpreter Python 3.
Zamiast Quizu tym razem efektami będzie kilka plików z zadań, które wg wytycznych wrzucicie do inbox.



Python w pigułce

Interpreter (vs. kompilator)

Tekst programu jest wykonywany i tłumaczony na kod maszynowy „w locie”. Wadą jest to, że niektóre błędy ujawnią się dopiero w trakcie działania programu, gdy nastąpi próba wykonania wadliwego fragmentu kodu.

Zaletą to dla nas oszczędność czasu – pliki źródłowe .py są edytowane i przechowywane bezpośrednio w pamięci Flash Pico – nie trzeba na nowo programować MCU.

Typowanie dynamiczne (vs. statyczne w C/C++)

Nie ma potrzeby deklarowania typów zmiennych (są sprawdzane w czasie wykonania ang. runtime, i mogą się zmieniać w trakcie działania programu).

```
x = "test"
```

```
x = 123
```

Podobnie jak w C, można jawnie rzutować typy, np. `x = int("123")`

Garbage collector

Zarządzanie pamięcią jest automatyczne.

Składnia

Definiowanie bloków kodu jest za pomocą wcięć, a nie { } znanych z C.
Brak znaków ; po każdej instrukcji

W przykładzie obok widać jak się definiuje funkcję, jak wygląda instrukcja warunkowa if .. else oraz pokazany jest jeden ze sposobów formatowania tekstów.

Zwróć uwagę na to, że napisy można definiować przy pomocy apostrofu ' jak i znaków cudzysłów ".

Literka f""" przed napisem jest wygodnym sposobem wstrzykiwania wartości zmiennych do napisu.

```
1 def zaliczenie(nr):  
2     if nr % 2 == 0:  
3         print(f"{nr} oblał.")  
4     else:  
5         print(f'{nr:09} zdał.')
```

```
7 zaliczenie(123456)  
8 zaliczenie(123455)
```

Shell x

```
>>> %Run syntax.py  
123456 oblał.  
000123455 zdał.
```

Konsola tekstowa (micropython, REPL)

Nie potrzeba konfigurować UART, komunikacja jest przez REPL

```
1 import select, sys, utime
2
3 def cmd_help(args):
4     print(f'Help me! args = {args}')
5
6 def cmd_df(args):
7     print(f'DanceFloor; args = {args}')
8     print(f'r={args[0]} g={args[1]} b={args[2]}')
9
10 def process_command(line):
11     if len(line) == 0: return
12     args = line.split()
13     if args[0] == 'help': cmd_help(args[1:])
14     elif args[0] == 'df': cmd_df(args[1:])
15     else:
16         print('Invalid command.')
17
18 while True:
19     if select.select([sys.stdin],[],[],0)[0]:
20         process_command(sys.stdin.readline().rstrip())
21     utime.sleep_ms(100)
```

Shell ×

```
>>> %Run -c $EDITOR_CONTENT
```

```
MPY: soft reboot
```

```
help
```

```
Help me! args = []
```

```
df 255 100 0
```

```
DanceFloor; args = ['255', '100', '0']
```

```
r=255 g=100 b=0
```

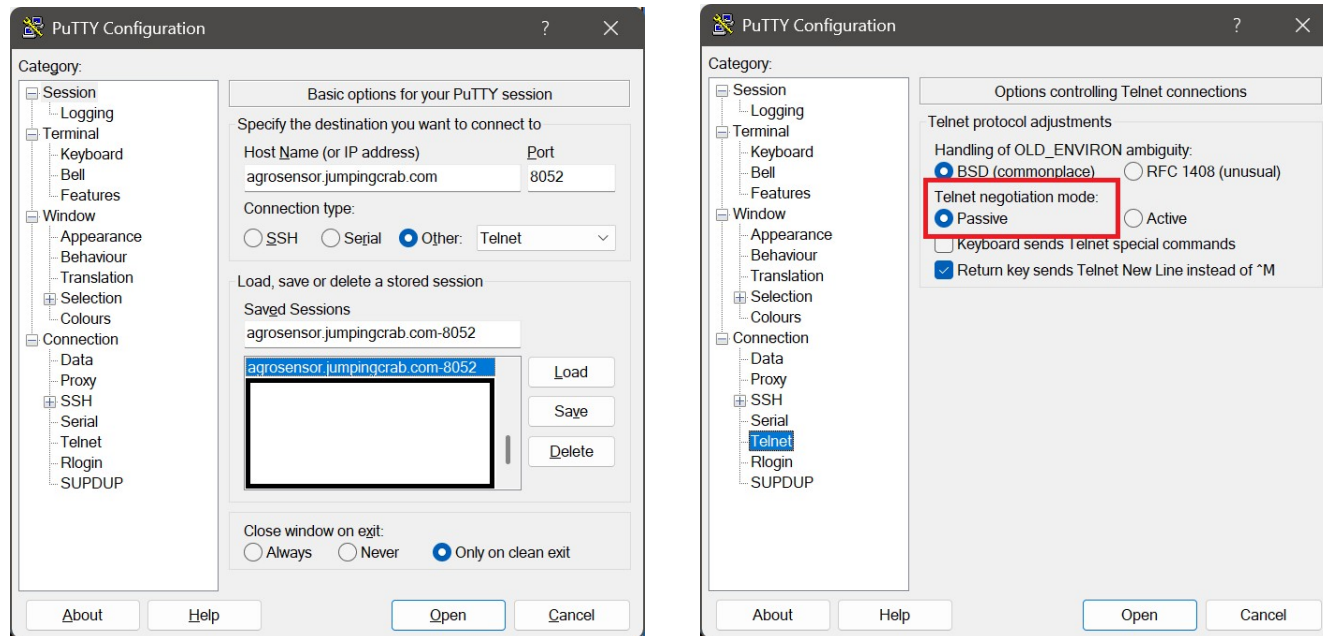
Konsola tekstowa (micropython, UART)

Bardzo podobne rozwiązanie do tego, którego używaliśmy w C

```
1 import utime
2 from machine import UART, Pin
3
4 uart_bytes = bytearray()
5 uart = UART(0, baudrate=115200, tx=Pin(0), rx=Pin(1))
6 uart.init(bits=8, parity=None, stop=1)
7
8 def send(data):
9     uart.write(data)
10    utime.sleep(0.1)
11
12 def process_command(line):
13     if len(line) == 0:
14         uart.write('empty\r\n')
15         return
16     args = line.split()
17     uart.write(f'{line}\r\n')
18
19 while True:
20     if uart.any():
21         data = uart.read()
22         for c in data:
23             if c >= 32:
24                 uart_bytes.append(c)
25             elif c == 13:
26                 try:
27                     process_command(uart_bytes.decode('UTF-8'))
28                 except: pass
29                 uart_bytes = bytearray()
30     else:
31         utime.sleep_ms(10)
```

Zadanie 0. SSH/PuTTY, TCP, PLAIN TEXT

Możesz użyć komendy telnet (Linux) lub np. programu PuTTY – pamiętaj o wybraniu trybu passive, connection type: Telnet (a nie SSH), port: 8052, adres jak wyżej.



Protokół komunikacji jest następujący:

"login NRALBUMU\r\n" - logowanie do serwera.

"cmd1\r\n" - po zalogowaniu i wysłaniu tej komendy otrzymasz ciąg znaków, który powinieneś wykonać jako komendę na BeamKit (teraz tylko wyświetli się napis w konsoli).

"quit\r\n" - kończy połączenie.


```
pi@kurnik:~ $ telnet agrosensor.jumpingcrab.com 8052
Trying 83.29.60.184...
Connected to agrosensor.jumpingcrab.com.
Escape character is '^]'.
login 123456
Hello, 123456
cmd1
df 4AEC29
cmd1
df CDBAAB
quit
Goodbye!
Connection closed by foreign host.
pi@kurnik:~ $
```

Przykładowy przebieg wymiany danych z serwerem.

Wyniki: plik NRALBUMU_zad0.png (graficzny zrzut zawartości okna) lub NRALBUMU_zad0.txt (zrzut tekstowy).

Zadanie 1. Python, PLAIN TEXT

Utwórz gniazdo TCP, połącz się z serwerem `agrosensor.jumpingcrab.com:8052` (uwaga, inny port niż dla HTTP).

Serwer TCP celowo dodaje sekundowe opóźnienie po wysłaniu odpowiedzi - chodzi o zabezpieczenie przed zbyt częstym wysyłaniem zapytań ze strony klienta.

Wyniki: `NRALBUMU_zad1.py` oraz zrzut ekranu programu.

```
1 import socket
2 SERVER_ADDR = 'agrosensor.jumpingcrab.com'
3 SERVER_PORT = 8052
4 DATA1 = "login 123456\r\n"
5 DATA2 = "cmd1\r\n"
6 DATA3 = "quit\r\n"
7
8 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
9     s.connect((SERVER_ADDR, SERVER_PORT))
10
11     s.sendall(DATA1.encode('ascii'))
12     response = s.recv(512).decode('ascii').strip()
13     print(f'{DATA1.strip()} ==> {response}')
14
15     s.sendall(DATA2.encode('ascii'))
16     response = s.recv(512).decode('ascii').strip()
17     print(f'{DATA2.strip()} ==> {response}')
18
19     s.sendall(DATA3.encode('ascii'))
20     response = s.recv(512).decode('ascii').strip()
21     print(f'{DATA3.strip()} ==> {response}')
```

Shell ×

```
>>> %Run zad1.py
```

```
login 123456 ==> Hello, 123456
```

```
cmd1 ==> df CDBAAB
```

```
quit ==> Goodbye!
```

```
>>>
```

Zadanie 2. Python, JSON

<http://agrosensor.jumpingcrab.com:8051/swa10-task1.php>

Przykładowa odpowiedź serwera:

```
{  
    "temperature":{"value":-20,"unit":"&deg;C"},  
    "pressure":{"value":1011,"unit":"hPa"},  
    "city":"Wroclaw"  
}
```

```
{"temperature":{"value":-20,"unit":"°C"},"pressure":{"value":1011,"unit":"hPa"},"city":"Wroclaw"}
```

Użyj wcześniejszego kodu TCP i „ręcznie” zaprogramuj metodę GET (wystarczy protokół HTTP/1.0). Nie używaj do tego gotowych bibliotek HTTP z Pythona.

Odebrane dane są w formacie JSON, na razie po prostu zapisz je w pliku.

Wyniki: pliki NRALBUMU_zad2.py oraz odpowiedź serwera NRALBUMU_zad2.json

Można to rozwiązać tak:

```
1 import socket
2 SERVER_ADDR = 'agrosensor.jumpingcrab.com'
3 SERVER_PORT = 8051
4 DATA1 = "GET /swa10-task1.php\r\n\r\n"
5
6 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
7     s.connect((SERVER_ADDR, SERVER_PORT))
8     s.sendall(DATA1.encode('ascii'))
9     response = s.recv(512).decode('ascii').strip()
10    print(f'{DATA1.strip()} ==> {response}')
```

Shell ×

```
>>> %Run zad2.py
```

```
GET /swa10-task1.php ==> {"temperature":{"value":-12,
"unit":"&deg;C"},"pressure":{"value":1005,"unit":"hPa
"},"city":"Wroclaw"}
```

```
>>>
```


A tak wygląda rozwiązanie z obsługą wyjątków, plików oraz protokołem HTTP/1.1.

Uwaga, z HTTP/1.1 należy poprawić kod, aby w pliku .json nie były zapisane nagłówki http.

```
1 import socket
2 SERVER_ADDR = 'agrosensor.jumpingcrab.com'
3 SERVER_PORT = 8051
4 # HTTP/1.0
5 # DATA1 = "GET /swa10-task1.php\r\n\r\n"
6 # HTTP/1.1
7 DATA1 = ("GET /swa10-task1.php HTTP/1.1\r\n"
8          "Host:agrosensor.jumpingcrab.com\r\n"
9          "\r\n")
10 try:
11     with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
12         s.settimeout(10) # zwiększ gdy sieć/serwer przeciążone
13         s.connect((SERVER_ADDR, SERVER_PORT))
14         s.sendall(DATA1.encode('ascii'))
15         response = s.recv(1024).decode('ascii')
16         print(f'{DATA1.strip()} ==> {response.strip()}')
17         f = open('zad2.json', 'w')
18         f.write(response)
19         f.close()
20 except socket.timeout:
21     print("socket.timeout")
22 except ConnectionRefusedError:
23     print("ConnectionRefusedError")
24 except socket.error as e:
25     print(f"socket.error: {e}")
26 except Exception as e:
27     print(f"Inny błąd: {e}")
```

Shell x

>>> %Run zad2.py

```
GET /swa10-task1.php HTTP/1.1
Host:agrosensor.jumpingcrab.com ==> HTTP/1.1 200 OK
Date: Wed, 04 Jun 2025 11:32:55 GMT
Server: Apache/2.4.29 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 101
Content-Type: text/html; charset=UTF-8
```

```
{"temperature":{"value":-12,"unit":"&deg;C"},"pressure":{"value":1030,"unit":"hPa"},"city":"Wroclaw"}
```

Zadanie 3. Python, urllib

Tym razem nie będziemy ręcznie składać GET i dekodować nagłówków, użyjemy urllib wbudowanej w Python.

```
1 import urllib.request
2 import urllib.error
3
4 URL = 'http://agrosensor.jumpingcrab.com:8051/swa10-task3.php'
5
6 try:
7     with urllib.request.urlopen(URL, timeout=10) as response:
8         print(f"Status odpowiedzi: {response.status} {response.reason}")
9         if response.status == 200:
10             content_bytes = response.read()
11             charset = response.headers.get_content_charset()
12             if charset is None:
13                 charset = 'utf-8' # Domyślne kodowanie
14             content_string = content_bytes.decode(charset)
15             print(content_string[:300]) # pierwsze 300 znaków
16         else:
17             print(f"Nie udało się pobrać danych. Kod błędu: {response.status}")
18 except urllib.error.HTTPError as e:
19     print(f"Błąd HTTP: {e.code} {e.reason}") # np. 404 Not Found
20 except urllib.error.URLError as e:
21     print(f"Błąd URL: {e.reason}")
22 except TimeoutError:
23     print("TimeoutError")
24 except Exception as e:
25     print(f"Inny błąd: {e}")
```

Shell ×

```
>>> %Run zad3.py
Status odpowiedzi: 200 OK
{"temperature":{"value":-6,"unit":"&deg;C"},"pressure":{"value":953,"
unit":"hPa"},"city":"Wroclaw"}
>>>
```

Wyniki: pliki NRALBUMU_zad3.py oraz odpowiedź serwera NRALBUMU_zad3.json

Zadanie 4. Biblioteka 'requests'

Domyślnie jej nie ma – pojawi się błąd:

```
>>> %Run zad4.py
Traceback (most recent call last):
  File "D:\OneDrive\PWr\embedded\dydaktyka\wykłady\SWA-W10\zad4.py",
  line 1, in <module>
    import requests
ModuleNotFoundError: No module named 'requests'
```

Aby zainstalować w Thonnym, wybierz Tools → Open system shell...

a następnie **pip install requests** :

```
D:\OneDrive\PWr\embedded\dydaktyka\wykłady\SWA-W10>pip install requests
Collecting requests
  Downloading requests-2.32.3-py3-none-any.whl (64 kB)
    _____ 64.9/64.9 kB 1.2 MB/s eta 0:00:00
Collecting certifi>=2017.4.17
  Downloading certifi-2025.4.26-py3-none-any.whl (159 kB)
    _____ 159.6/159.6 kB 4.7 MB/s eta 0:00:00
Collecting urllib3<3,>=1.21.1
  Downloading urllib3-2.4.0-py3-none-any.whl (128 kB)
    _____ 128.7/128.7 kB ? eta 0:00:00
Collecting charset-normalizer<4,>=2
  Downloading charset_normalizer-3.4.2-cp310-cp310-win_amd64.whl (105 kB)
    _____ 105.8/105.8 kB ? eta 0:00:00
Collecting idna<4,>=2.5
  Downloading idna-3.10-py3-none-any.whl (70 kB)
    _____ 70.4/70.4 kB ? eta 0:00:00
Installing collected packages: urllib3, idna, charset-normalizer, certifi, requests
Successfully installed certifi-2025.4.26 charset-normalizer-3.4.2 idna-3.10 requests-2.32.3 urllib3-2.4.0
```

```
1 import requests
2 URL = 'http://agrosensor.jumpingcrab.com:8051/swa10-task3.php'
3 try:
4     response = requests.get(URL, timeout=10)
5     response.raise_for_status() # wyjątek dla 4xx lub 5xx
6     print(f"Status odpowiedzi: {response.status_code}")
7     html_content = response.text
8     print(response.text[:500]) # pierwsze 500 znaków
9 except requests.exceptions.HTTPError as e:
10     print(f"HTTPError: {e}")
11 except requests.exceptions.ConnectionError as e:
12     print(f"ConnectionError: {e}")
13 except requests.exceptions.Timeout as e:
14     print(f"Timeout: {e}")
15 except requests.exceptions.RequestException as e:
16     print(f"RequestException: {e}")
17 except Exception as e:
18     print(f"Exception: {e}")
```

Shell ×

```
>>> %Run zad4.py
```

```
Status odpowiedzi: 200
```

```
{"temperature":{"value":14,"unit":"&deg;C"},"pressure":{"value":985,"unit":"hPa"},"city":"Wroclaw"}
```

```
>>>
```

Wyniki: pliki NRALBUMU_zad4.py oraz odpowiedź serwera NRALBUMU_zad4.json

Biblioteka 'urequests' (MicroPython)

BeamKit – prawie tak samo

- użyj urequests (dodatkowego pakiet, wymaga doinstalowania)
- połącz się z WiFi

Ten fragment włącza WiFi:

```
import urequests, network, utime

sta = network.WLAN(network.STA_IF)
if not sta.isconnected():
    sta.active(True)
    sta.connect("SWA", "systemywbudowane")
    while not sta.isconnected():
        utime.sleep_ms(500)
print('Connected:', sta.ifconfig())
```

To zadanie będzie do wykonania na laborce w w piątek.

JSON Connor [<https://www.youtube.com/watch?v=6r9PpsaTunM>]

Zadanie 6. Używając requests, pobierz z serwera HTTP dane.

Zdekoduj te dane do formatu JSON. Wydobądź pole 'color' i przekształć do postaci liczb r,g,b.

```
1 import requests
2 #URL = 'http://agrosensor.jumpingcrab.com:8051/swa10-task5.php?color=FFFFFF'
3 URL = 'http://agrosensor.jumpingcrab.com:8051/swa10-task5.php'
4 try:
5     response = requests.get(URL, timeout = 10)
6     response.raise_for_status() # wyjątek dla 4xx lub 5xx
7     if response.status_code == 200:
8         json = response.json()
9         print(json)
10 except Exception as e:
11     print(f"Exception: {e}")
```

Shell ×

```
>>> %Run zad5.py
{'color': 'FFFFFF', 'date': '25-06-04', 'time': '13:39:21'}
>>>
```

Podanie argumentu color=xxxxxx w metdzie GET zapisze na serwerze nowy kolor. Bez tego argumentu wszyscy odczytają kolor zapamiętany na serwerze.


```
1 import requests
2 #URL = 'http://agrosensor.jumpingcrab.com:8051/swa10-task5.php?color=FFFFFF'
3 URL = 'http://agrosensor.jumpingcrab.com:8051/swa10-task5.php'
4 try:
5     response = requests.get(URL, timeout = 10)
6     response.raise_for_status() # wyjątek dla 4xx lub 5xx
7     if response.status_code == 200:
8         json = response.json()
9         #print(json)
10        r = int(json['color'][0:2], 16)
11        g = int(json['color'][2:4], 16)
12        b = int(json['color'][4:6], 16)
13        print(r, g, b)
14 except Exception as e:
15     print(f"Exception: {e}")
```

Shell ×

```
>>> %Run zad5.py
```

```
171 205 239
```

```
>>>
```

Wyniki: pliki NRALBUMU_zad6.py oraz odpowiedź serwera NRALBUMU_zad6.json

Zadanie 11. Jak 1., ale użyj BeamKit. Wyrzuć instrukcję with oraz szczegółowe wyjątki.

Wyniki:

```
>>> %Run -c $EDITOR_CONTENT
```

```
MPY: soft reboot
```

```
Connected: ('10.0.0.200', '255.255.255.0', '10.0.0.1', '8.8.8.8')
```

```
login 123456 ==> Hello, 123456
```

```
cmd1 ==> df 67C669
```

```
quit ==> Goodbye!
```

```
>>>
```

Zadanie 12. Jak 2., ale użyj BeamKit. Wyrzuć instrukcję with oraz szczegółowe wyjątki.

Plik zad2.json zostanie zapisany w pamięci Flash Pico!

Wyniki dla HTTP/1.0:

```
>>> %Run -c $EDITOR_CONTENT
```

```
MPY: soft reboot
```

```
Connected: ('10.0.0.200', '255.255.255.0', '10.0.0.1', '8.8.8.8')
```

```
GET /swa10-task1.php ==> {"temperature":{"value":30,"unit":"&deg;C"},  
"pressure":{"value":976,"unit":"hPa"},"city":"Wroclaw"}
```

```
>>>
```

Wyniki dla HTTP/1.1:

```
>>> %Run -c $EDITOR_CONTENT
```

```
MPY: soft reboot
```

```
Connected: ('10.0.0.200', '255.255.255.0', '10.0.0.1', '8.8.8.8')
```

```
GET /swa10-task1.php HTTP/1.1
```

```
Host:10.0.0.198 ==> HTTP/1.1 200 OK
```

```
Date: Thu, 05 Jun 2025 14:51:14 GMT
```

```
Server: Apache/2.4.62 (Raspbian)
```

```
Vary: Accept-Encoding
```

```
Content-Length: 100
```

```
Content-Type: text/html; charset=UTF-8
```

```
{"temperature":{"value":-1,"unit":"&deg;C"},"pressure":{"value":1004,"unit":"hPa"},"city":"Wroclaw"}
```

```
>>>
```


Zadanie 14. Jak 4., ale użyj BeamKit + urequests.

Wyniki:

```
MPY: soft reboot
Connected: ('10.0.0.200', '255.255.255.0', '10.0.0.1', '8.8.8.8')
content: {"temperature":{"value":9,"unit":"&deg;C"},"pressure":{"value":1040,"unit":"hPa"},"city":"Wroclaw"}
JSON: {'pressure': {'value': 1040, 'unit': 'hPa'}, 'temperature': {'value': 9, 'unit': '&deg;C'}, 'city': 'Wroclaw'}
>>>
```

Zadanie 16. Jak 6., ale użyj BeamKit + urequests.

Pamiętaj, aby dodać fragment kodu, który połączy się z WiFi w labie.

Jako adresu IP najlepiej użyj 10.0.0.198.

Zadanie 17. Dodaj obsługę podstawowych komend:

df XXXXXX – zapala cały df na dany kolor

connect – łączy się ze zdalnym serwerem (id, adres, port mogą być predefiniowane w kodzie źródłowym)

disconnect – zakończy połączenie z serwerem

weather – pobiera aktualną temperaturę i ciśnienie z serwera (i wyświetla w konsoli oraz na OLED)

oled wiersz tekst – wyświetla w wierszu 0/1/2/3 podany tekst

color – pobiera aktualnie modny kolor z serwera i aktualizuje df

tcp komenda argumenty – wysyła do serwera TCP:8053 komendę postaci 'cmd komenda argumenty' – serwer po jej odebraniu roześle ją do wszystkich połączonych klientów.

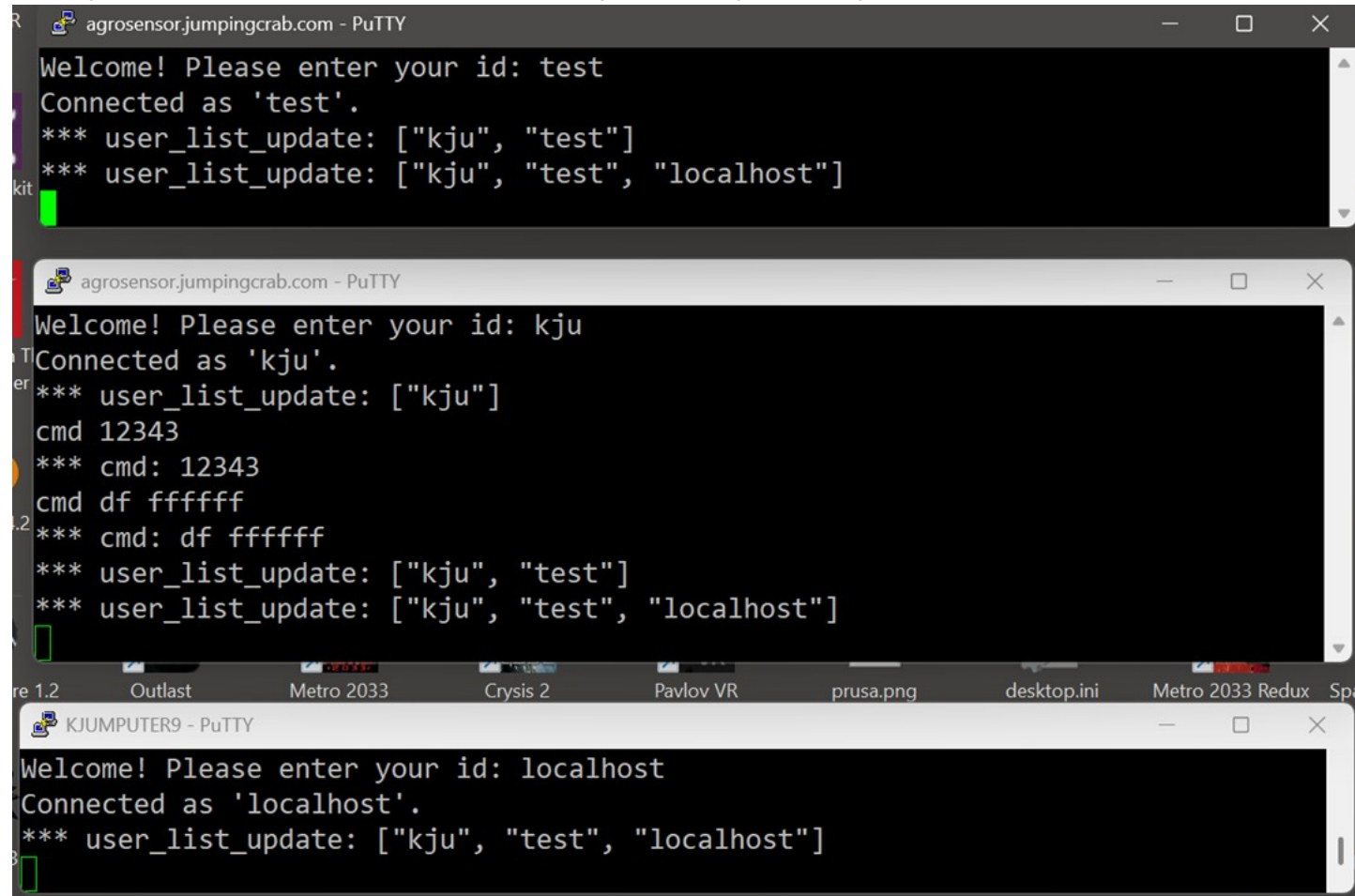
Wskazówka: przydatny będzie pakiet `_thread` , aby klient TCP działał w tle – w protokole 8053 występują niezapowiedziane komunikaty ze strony serwera. Trudniej nad tym zapanować w kodzie, ale nie trzeba ciągle odpytywać → porównaj z HTTP Push

Dodatkowo obsługa komunikatów z TCP:8053 postaci:

*** cmd: komenda argumenty

*** user_list_update: ["user1", "user2"]

Przykładowe scenariusz wymiany danych z serwerem 8053:



The image displays three stacked PuTTY terminal windows, each connected to the server `agrosensor.jumpingcrab.com`. The windows illustrate a sequence of data exchanges between a client and a server.

Top Window (User: test):

```
Welcome! Please enter your id: test
Connected as 'test'.
*** user_list_update: ["kju", "test"]
*** user_list_update: ["kju", "test", "localhost"]
```

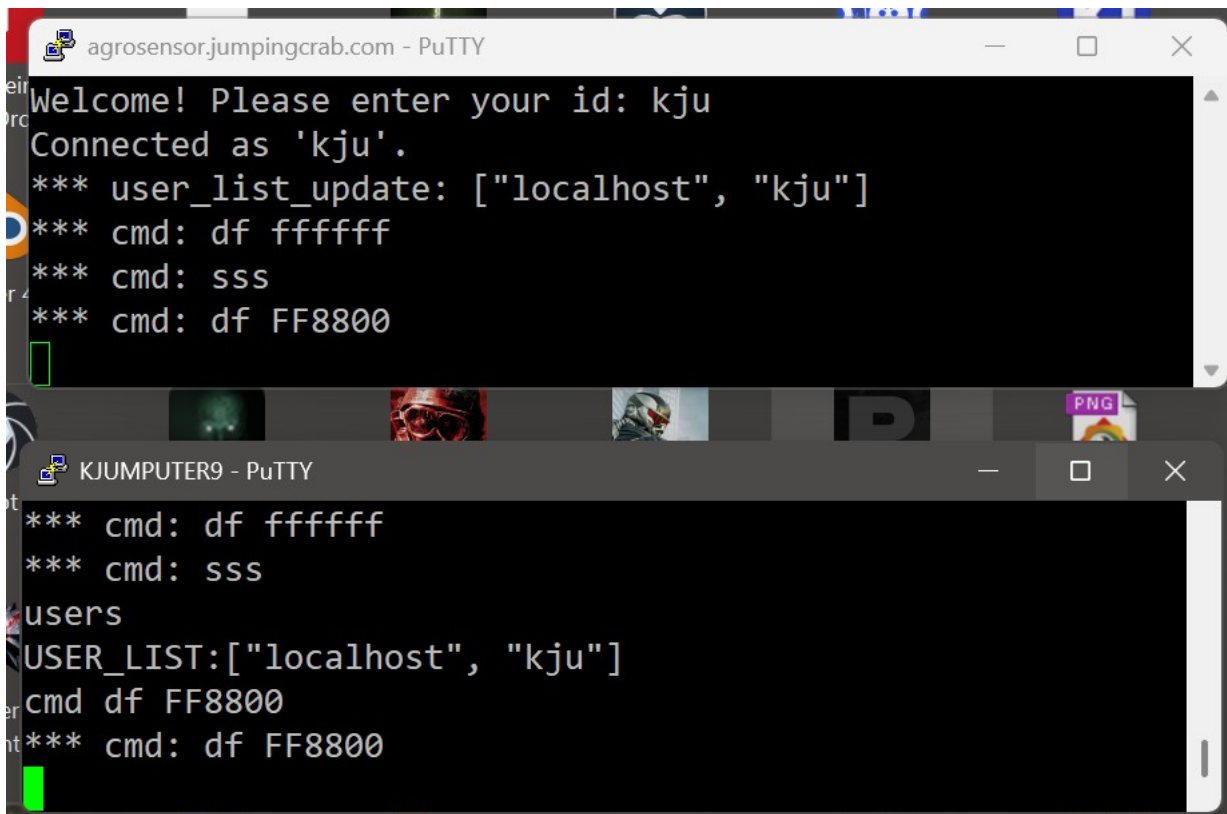
Middle Window (User: kju):

```
Welcome! Please enter your id: kju
Connected as 'kju'.
*** user_list_update: ["kju"]
cmd 12343
*** cmd: 12343
cmd df fffffff
*** cmd: df fffffff
*** user_list_update: ["kju", "test"]
*** user_list_update: ["kju", "test", "localhost"]
```

Bottom Window (User: localhost):

```
Welcome! Please enter your id: localhost
Connected as 'localhost'.
*** user_list_update: ["kju", "test", "localhost"]
```

The bottom window's title bar shows a list of open files: `re 1.2`, `Outlast`, `Metro 2033`, `Crysis 2`, `Pavlov VR`, `prusa.png`, `desktop.ini`, `Metro 2033 Redux`, and `Spa`.



```
agrosensor.jumpingcrab.com - PuTTY
Welcome! Please enter your id: kju
Connected as 'kju'.
*** user_list_update: ["localhost", "kju"]
*** cmd: df fffffff
*** cmd: sss
*** cmd: df FF8800

KJUMPUTER9 - PuTTY
*** cmd: df fffffff
*** cmd: sss
users
USER_LIST:["localhost", "kju"]
cmd df FF8800
*** cmd: df FF8800
```

Od strony MicroPython wybrane fragment kodu mogą wyglądać tak:

```
96 cmd_connect(None, CMD_SOURCE_CODE)
97 process_command("df 010101", CMD_SOURCE_CODE)
98 while True:
99     if select.select([sys.stdin], [], [], 0)[0]:
100         process_command(sys.stdin.readline().strip(), CMD_SOURCE_REPL)
101         utime.sleep_ms(100)
```

Po włączeniu programu nawiązywane jest połączenie oraz testowany jest lokalny df (minimalna jasność)

Prymitywny, ale działający dekodery komend tekstowych:

```
51 def process_command(cmd, source):
52     cmd = cmd.split(' ')
53     print(f'processing command: {cmd}')
54     if cmd[0] == 'close': cmd_close(cmd[1:], source)
55     elif cmd[0] == 'df': cmd_df(cmd[1:], source)
56     elif cmd[0] == 'quit': cmd_quit(cmd[1:], source)
```

Dodałem rozróżnianie, skąd pochodzi komenda (tzn.: konsola, serwer TCP, wywołanie w kodzie):

```
22 CMD_SOURCE_REPL = 1
23 CMD_SOURCE_TCP = 2
24 CMD_SOURCE_CODE = 3
```

Chodzi o to, aby złośliwi użytkownicy serwera nie wysyłali komend typu 'reset' albo 'quit' do wszystkich klientów, a właściwie aby klienci nie reagowali na wszystkie komendy z TCP.