**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ЛЬВІВСЬКА ПОЛІТЕХНІКА**



**АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ**

**КОМП’ЮТЕРНИХ СИСТЕМ**

Task5 Implement automated tests

Виконав :

Студент групи КІ-401

Онисько М.М.

Прийняв:

Федак П.Р

**2024**

**Завдання:**

1. Розробити автоматизовані тести для програмної частини та обладнання
2. Створити CI файл для автоматизованого запуску тестів
3. Оновити .yaml скрипт для GiHub щоб запускати тести на GitHub
4. Оновити Readme.md
5. Додати тег про нову версію
6. Злити створену гілку до develop

**Теоретичні відомості:**

Автоматизоване тестування передбачає використання інструменту автоматизації для виконання набору тестів. У той час як ручне тестування виконується людиною, що сидить перед комп’ютером, ретельно виконує всі етапи тестування.

Автоматизація ПЗ також може вводити тестові дані в систему, яку тестують, порівнювати очікувані та фактичні результати та генерувати детальні звіти про тестування. Однак воно вимагає значного вкладання коштів та ресурсів.

Цикл розробки вимагає багаторазового виконання одного й того ж набору тестів під час послідовності розробки. Використовуючи автоматизацію, можна написати набір тестів і відтворювати його повторно у разі необхідності. Як тільки набір тестів автоматизовано, втручання людини не потрібне. Також це допомагає поліпшити ROI (коефіцієнт окупності інвестицій). Метою автоматизації є скорочення кількості тестів, які потрібно запускати вручну, а не усунення ручного тестування в цілому.

Автоматизоване тестування програмного забезпечення є важливим з наступних причин:

* Ручне тестування усіх робочих процесів, усіх полів, усіх негативних сценаріїв вимагає багато часу та грошей.
* Доволі складно протестувати мультимовні сайти вручну.
* Автоматизація не вимагає втручання людини. Ви можете запустити автоматичний тест без нагляду (наприклад вночі).
* Автоматизація збільшує швидкість виконання тесту.
* Автоматизація допомагає збільшити покриття тестами (Test Coverage).
* Ручне тестування може бути нудним а, отже, веде до випадкових помилок.

**Модульне тестування**

Модульне тестування (англ. Unit testing) — це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яку може бути протестованою. У процедурному програмуванні модулем вважають окрему функцію або процедуру. В об'єктно-орієнтованому програмуванні — інтерфейс, клас. Модульні тести, або unit-тести, розробляються в процесі розробки програмістами та, іноді, тестувальниками білої скриньки (white-box testers).

Однією з головних проблем автоматизованого тестування є його трудомісткість: попри те, що воно дозволяє усунути частину рутинних операцій і прискорити виконання тестів, великі ресурси можуть витрачатися на оновлення самих тестів. Це відноситься до обох видів автоматизації. При рефакторінгу часто буває необхідно оновити і модульні тести, і зміна коду тестів може зайняти стільки ж часу, скільки і зміна основного коду. З іншого боку, при зміні інтерфейсу програми необхідно заново переписати всі тести, які пов'язані з оновленими вікнами, що при великій кількості тестів може відняти значні ресурси.

**Виконання**

Розробив тести, використовуючи фреймворк unittest мови Python

**Код тестів для HW частини**

|  |
| --- |
| import unittest  import serial  import json  import time  SERIAL\_PORT = 'COM3' # Change to your COM\_PORT  BAUD\_RATE = 9600  class TicTacToeArduinoTests(unittest.TestCase):  @classmethod  def setUpClass(cls):  cls.ser = serial.Serial(SERIAL\_PORT, BAUD\_RATE, timeout=1)  time.sleep(2)  @classmethod  def tearDownClass(cls):  cls.ser.close()  def send\_command(self, command\_dict):  self.ser.write((json.dumps(command\_dict) + '\n').encode())  time.sleep(0.5)  def receive\_response(self):  if self.ser.in\_waiting > 0:  line = self.ser.readline().decode().strip()  try:  return json.loads(line)  except json.JSONDecodeError:  return None  return None  def test\_initialize\_board(self):  self.send\_command({"command": "RESET"})  response1 = self.receive\_response()  response2 = self.receive\_response()  self.assertIsNotNone(response2)  if response2["type"] == "game\_status":  self.assertEqual(response2["message"], "Game reset.")  response2 = self.receive\_response()  self.assertEqual(response2["type"], "board")  board\_state = response2.get("board", [])  for row in board\_state:  for cell in row:  self.assertEqual(cell, " ")  def test\_make\_valid\_move(self):  self.send\_command({"command": "RESET"})  self.receive\_response()  self.receive\_response()  self.send\_command({"command": "MOVE", "row": 0, "col": 0})  response = self.receive\_response()  self.assertEqual(response["type"], "board")  board\_state = response.get("board", [])  self.assertEqual(board\_state[0][0], "X")  def test\_make\_invalid\_move(self):  self.send\_command({"command": "RESET"})  self.receive\_response()  self.receive\_response()  self.send\_command({"command": "MOVE", "row": 0, "col": 0})  self.receive\_response()  self.send\_command({"command": "MOVE", "row": 0, "col": 0})  response = self.receive\_response()  self.assertIsNotNone(response)  if response["type"] == "error":  self.assertEqual(response["message"], "Invalid move.")  else:  self.assertEqual(response["type"], "board")  def test\_check\_win(self):  self.send\_command({"command": "RESET"})  self.receive\_response()  self.receive\_response()  moves = [(0, 0), (1, 0), (0, 1), (1, 1), (0, 2)]  for row, col in moves:  self.send\_command({"command": "MOVE", "row": row, "col": col})  self.receive\_response()  response = self.receive\_response()  self.assertEqual(response["type"], "win\_status")  self.assertEqual(response["message"], "Player X wins!")  def test\_draw(self):  self.send\_command({"command": "RESET"})  self.receive\_response()  self.receive\_response()  moves = [  (0, 0), (0, 1), (0, 2),  (1, 1), (1, 0), (1, 2),  (2, 1), (2, 0), (2, 2)  ]  for row, col in moves:  self.send\_command({"command": "MOVE", "row": row, "col": col})  self.receive\_response()  response = self.receive\_response()  self.assertEqual(response["type"], "win\_status")  self.assertEqual(response["message"], "It's a draw!")  def test\_game\_mode\_switch(self):  """Test switching between different game modes."""  self.send\_command({"command": "MODE", "mode": 1})  responses = {"game\_mode": False, "game\_status": False, "board": False}  for \_ in range(5):  response = self.receive\_response()  if response:  response\_type = response["type"]  if response\_type == "game\_mode":  responses["game\_mode"] = True  self.assertIn("Game mode set to 1", response["message"])  elif response\_type == "game\_status":  responses["game\_status"] = True  self.assertEqual(response["message"], "Game reset.")  if all(responses.values()):  break  # Change mode to 2 (AI vs AI)  self.send\_command({"command": "MODE", "mode": 2})  responses = {"game\_mode": False, "game\_status": False, "board": False}  for \_ in range(5):  response = self.receive\_response()  if response:  response\_type = response["type"]  if response\_type == "game\_mode":  responses["game\_mode"] = True  self.assertIn("Game mode set to 2", response["message"])  elif response\_type == "game\_status":  responses["game\_status"] = True  self.assertEqual(response["message"], "Game reset.")  elif response\_type == "board":  responses["board"] = True  board\_state = response["board"]  for row in board\_state:  for cell in row:  self.assertEqual(cell, " ")  if all(responses.values()):  break  def test\_handle\_ai\_vs\_ai(self):  self.send\_command({"command": "MODE", "mode": 2})  self.receive\_response()  self.receive\_response()  while True:  response = self.receive\_response()  if response and response["type"] == "win\_status":  self.assertIn(response["message"], ["Player X wins!", "Player O wins!", "It's a draw!"])  break  if \_\_name\_\_ == '\_\_main\_\_':  unittest.main() |

**Код тестів для клієнтської частини**

|  |
| --- |
| # import logging  #  # logger = logging.getLogger('test\_application')  # logger.setLevel(logging.DEBUG)  # fh = logging.FileHandler('test.log')  # fh.setLevel(logging.DEBUG)  # logger.addHandler(fh)  import unittest  from unittest.mock import MagicMock, patch  from tkinter import Tk  from io import StringIO  from main import UARTCommunication, update\_game\_board, send\_move, set\_mode, reset\_game, auto\_receive  import tkinter as tk  from tkinter import scrolledtext  class TestUARTCommunication(unittest.TestCase):  def setUp(self):  self.uart = UARTCommunication()  @patch('serial.tools.list\_ports.comports')  def test\_list\_ports(self, mock\_comports):  mock\_comports.return\_value = [MagicMock(device="COM3"), MagicMock(device="COM4")]  ports = self.uart.list\_ports()  self.assertEqual(ports, ["COM3", "COM4"])  @patch('serial.Serial')  def test\_open\_port\_success(self, mock\_serial):  mock\_serial.return\_value = MagicMock(is\_open=True)  status = self.uart.open\_port("COM3")  self.assertEqual(status, "Connected to COM3")  self.assertTrue(self.uart.ser.is\_open)  @patch('serial.Serial')  def test\_open\_port\_failure(self, mock\_serial):  mock\_serial.side\_effect = Exception("Port error")  status = self.uart.open\_port("COM5")  self.assertEqual(status, "Error: Port error")  self.assertIsNone(self.uart.ser)  @patch('serial.Serial')  def test\_send\_message\_success(self, mock\_serial):  mock\_serial.return\_value = MagicMock(is\_open=True)  self.uart.ser = mock\_serial()  result = self.uart.send\_message({"command": "MOVE", "row": 0, "col": 1})  self.assertIn("Sent:", result)  @patch('serial.Serial')  def test\_send\_message\_failure(self, mock\_serial):  mock\_serial.return\_value = MagicMock(is\_open=False)  self.uart.ser = mock\_serial()  result = self.uart.send\_message({"command": "MOVE", "row": 0, "col": 1})  self.assertEqual(result, "Port not opened")  @patch('serial.Serial')  def test\_receive\_message\_success(self, mock\_serial):  mock\_serial.return\_value = MagicMock(is\_open=True, in\_waiting=1)  mock\_serial().readline.return\_value = b'{"command": "MOVE", "row": 0, "col": 1}\n'  self.uart.ser = mock\_serial()  message = self.uart.receive\_message()  self.assertEqual(message, {"command": "MOVE", "row": 0, "col": 1})  @patch('serial.Serial')  def test\_receive\_message\_invalid\_json(self, mock\_serial):  mock\_serial.return\_value = MagicMock(is\_open=True, in\_waiting=1)  mock\_serial().readline.return\_value = b'{"command": "MOVE", "row": 0, "col": }\n'  self.uart.ser = mock\_serial()  message = self.uart.receive\_message()  self.assertEqual(message, "Error: Invalid JSON received")  @patch('serial.Serial')  def test\_receive\_message\_no\_data(self, mock\_serial):  mock\_serial.return\_value = MagicMock(is\_open=True, in\_waiting=0)  self.uart.ser = mock\_serial()  message = self.uart.receive\_message()  self.assertEqual(message, "Port not opened")  class TestGameFunctions(unittest.TestCase):  def setUp(self):  self.uart = UARTCommunication() # Ensure uart is set up for each test  def test\_update\_game\_board(self):  root = Tk()  buttons = [[tk.Button(root, text=" ") for \_ in range(3)] for \_ in range(3)]  board = [["X", "O", "X"], ["O", "X", "O"], ["X", "O", "X"]]  update\_game\_board(board, buttons)  for i in range(3):  for j in range(3):  self.assertEqual(buttons[i][j]["text"], board[i][j])  root.destroy()  @patch.object(UARTCommunication, 'send\_message')  def test\_send\_move(self, mock\_send\_message):  send\_move(self.uart, 1, 1)  mock\_send\_message.assert\_called\_with({"command": "MOVE", "row": 1, "col": 1})  @patch.object(UARTCommunication, 'send\_message')  def test\_set\_mode(self, mock\_send\_message):  set\_mode(self.uart, 1)  mock\_send\_message.assert\_called\_with({"command": "MODE", "mode": 1})  @patch.object(UARTCommunication, 'send\_message')  def test\_reset\_game(self, mock\_send\_message):  reset\_game(self.uart)  mock\_send\_message.assert\_called\_with({"command": "RESET"})  @patch('serial.Serial')  def test\_auto\_receive\_no\_data(self, mock\_serial):  mock\_serial.return\_value = MagicMock(is\_open=True, in\_waiting=2)  self.uart.ser = mock\_serial()  root = Tk()  buttons = [[tk.Button(root, text=" ") for \_ in range(3)] for \_ in range(3)]  output\_text = scrolledtext.ScrolledText(root, width=50, height=10)  # Simulate no data received  mock\_serial().readline.return\_value = b''  auto\_receive(self.uart, buttons, output\_text, root)  # Check if no board update happens  for i in range(3):  for j in range(3):  self.assertEqual(buttons[i][j]["text"], " ")  root.destroy()  def test\_uart\_initialization(self):  uart = UARTCommunication()  self.assertIsNone(uart.ser)  @patch('serial.Serial')  def test\_auto\_receive\_valid\_response(self, mock\_serial):  mock\_serial.return\_value = MagicMock(is\_open=True, in\_waiting=1)  mock\_serial().readline.return\_value = b'{"board": [["X", "O", "X"], ["O", "X", "O"], ["X", "O", "X"]]}'  self.uart.ser = mock\_serial()  root = Tk()  buttons = [[tk.Button(root, text=" ") for \_ in range(3)] for \_ in range(3)]  output\_text = scrolledtext.ScrolledText(root, width=50, height=10)  # Simulate receiving a valid game board response  auto\_receive(self.uart, buttons, output\_text, root)  # Check if the board was updated correctly  for i in range(3):  for j in range(3):  self.assertEqual(buttons[i][j]["text"], ["X", "O", "X", "O", "X", "O", "X", "O", "X"][i \* 3 + j])  root.destroy()  @patch('serial.Serial')  def test\_auto\_receive\_invalid\_json(self, mock\_serial):  mock\_serial.return\_value = MagicMock(is\_open=True, in\_waiting=1)  mock\_serial().readline.return\_value = b'{"board": [["X", "O", "X"], ["O", "X", "O"]]}'  self.uart.ser = mock\_serial()  root = Tk()  buttons = [[tk.Button(root, text=" ") for \_ in range(3)] for \_ in range(3)]  output\_text = scrolledtext.ScrolledText(root, width=50, height=10)  # Simulate receiving an invalid game board response  auto\_receive(self.uart, buttons, output\_text, root)  # Check if error message is displayed  self.assertIn("Error:", output\_text.get("1.0", tk.END))  root.destroy()  if \_\_name\_\_ == '\_\_main\_\_':  unittest.main() |

**Автоматизація**

**DoxygenCreateWindows.ps – Лістинг СІ скрипту для запуску тестів**

|  |
| --- |
| # PowerShell Script to Install Doxygen and Generate Documentation  # Define paths  $doxygenInstallerUrl = "https://doxygen.nl/files/doxygen-1.12.0-setup.exe" # Replace with the latest version if needed  $doxygenInstallerPath = "$env:TEMP\doxygen-setup.exe"  $projectDir = Join-Path -Path $PSScriptRoot -ChildPath ".." # Replace with the path to your project  $outputDir = "$projectDir\docs" # Path for generated documentation  # Step 1: Check if Doxygen is installed  Write-Output "Checking if Doxygen is installed..."  $doxygenPath = (Get-Command "doxygen" -ErrorAction SilentlyContinue).Source  if (-not $doxygenPath) {  Write-Output "Doxygen not found. Downloading and installing Doxygen..."  # Download Doxygen installer  Invoke-WebRequest -Uri $doxygenInstallerUrl -OutFile $doxygenInstallerPath -UseBasicParsing  # Run the installer silently  Start-Process -FilePath $doxygenInstallerPath -ArgumentList "/S" -Wait  # Confirm installation  $doxygenPath = (Get-Command "doxygen" -ErrorAction SilentlyContinue).Source  if (-not $doxygenPath) {  Write-Output "Doxygen installation failed. Please install it manually."  exit 1  }  # Add Doxygen to PATH  $doxygenPath = "C:\Program Files\doxygen\bin" # Default installation path, adjust if different  [System.Environment]::SetEnvironmentVariable("Path", $env:Path + ";$doxygenPath", [System.EnvironmentVariableTarget]::Machine)  Write-Output "Doxygen installed successfully."  } else {  Write-Output "Doxygen is already installed at $doxygenPath."  }  # Step 2: Create Doxygen configuration file if not exists  $doxyfilePath = "$projectDir\Doxyfile"  if (-not (Test-Path $doxyfilePath)) {  Write-Output "Generating Doxygen configuration file..."  Start-Process -FilePath "doxygen" -ArgumentList "-g $doxyfilePath" -Wait  }  # Step 3: Update configuration file for your project settings  (Get-Content $doxyfilePath) -replace 'OUTPUT\_DIRECTORY.\*', "OUTPUT\_DIRECTORY = $outputDir" | Set-Content $doxyfilePath  (Get-Content $doxyfilePath) -replace 'INPUT.\*', "INPUT = $projectDir" | Set-Content $doxyfilePath  (Get-Content $doxyfilePath) -replace 'RECURSIVE.\*', "RECURSIVE = YES" | Set-Content $doxyfilePath  # Step 4: Run Doxygen to generate documentation  Write-Output "Generating documentation..."  Start-Process -FilePath "doxygen" -ArgumentList "$doxyfilePath" -Wait  Write-Output "Documentation generation complete. Output available at $outputDir." |

**Github\_CI.yaml – Лістинг СІ скрипту для запуску GitHub Actions**

|  |
| --- |
| name: CI Workflow  on:  push:  branches:  - develop  - feature/develop/task5  pull\_request:  branches:  - develop  jobs:  build:  runs-on: windows-latest  steps:  # Checkout the repository  - name: Checkout code  uses: actions/checkout@v3  # Set up Python environment  - name: Set up Python 3.x  uses: actions/setup-python@v4  with:  python-version: '3.x'  # Install Python dependencies  - name: Install dependencies  run: |  python -m pip install --upgrade pip  pip install -r TicTacToeSWPart/requirements.txt  pip install pytest  # Install Arduino CLI on Windows  - name: Set up Arduino CLI  run: |  Invoke-WebRequest -Uri https://downloads.arduino.cc/arduino-cli/arduino-cli\_latest\_Windows\_64bit.zip -OutFile arduino-cli.zip  Expand-Archive -Path arduino-cli.zip -DestinationPath $Env:USERPROFILE\arduino-cli  $ArduinoCLIPath = "$Env:USERPROFILE\arduino-cli\arduino-cli.exe"  $Env:PATH += ";$Env:USERPROFILE\arduino-cli"  & $ArduinoCLIPath config init  # Install Arduino AVR core  - name: Install Arduino AVR core  run: |  $ArduinoCLIPath = "$Env:USERPROFILE\arduino-cli\arduino-cli.exe"  & $ArduinoCLIPath core update-index  & $ArduinoCLIPath core install arduino:avr  # Install ArduinoJson library  - name: Install ArduinoJson library  run: |  $ArduinoCLIPath = "$Env:USERPROFILE\arduino-cli\arduino-cli.exe"  & $ArduinoCLIPath lib install ArduinoJson  # Compile Arduino Sketch for Arduino Nano (Old Bootloader)  - name: Compile Arduino Sketch  run: |  $ArduinoCLIPath = "$Env:USERPROFILE\arduino-cli\arduino-cli.exe"  mkdir build  & $ArduinoCLIPath compile --fqbn arduino:avr:nano:cpu=atmega328old --output-dir build HWPart/TicTacToe/TicTacToe.ino  # Run tests using the PowerShell script  - name: Run Tests  run: |  PowerShell -ExecutionPolicy Bypass -File ./CI/RunTestsWindows.ps1  # Upload compiled binaries as artifacts  - name: Upload binaries  uses: actions/upload-artifact@v3  with:  name: compiled-files  path: build/\*.bin  # Collect test results as artifacts  - name: Upload test reports  uses: actions/upload-artifact@v3  with:  name: test-reports  path: TicTacToeSWPart/test-reports/\*.xml |

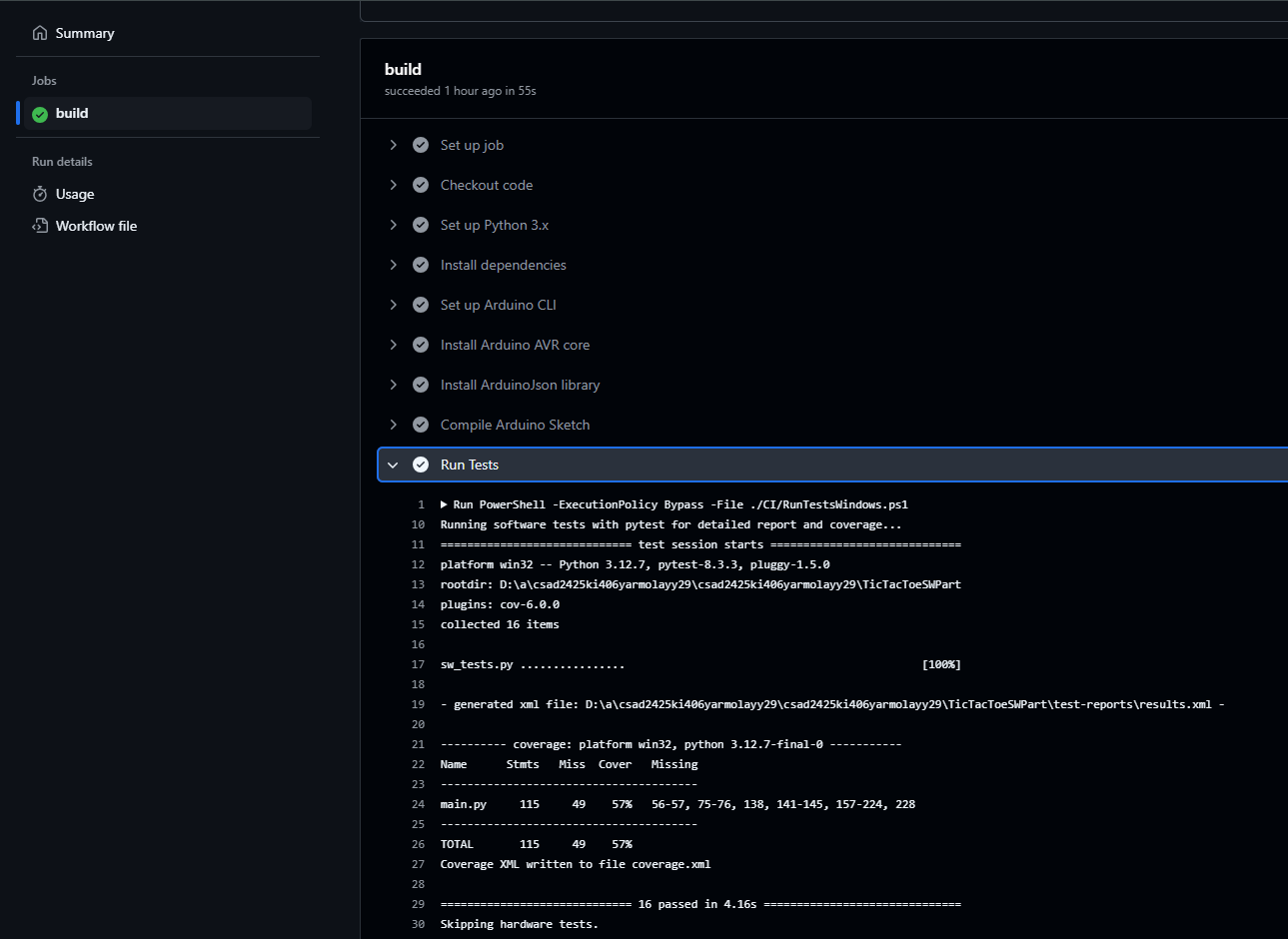


Рис.2 – Виконання GitHub Actions

**Висновки**

На лабораторній роботі я зробив тести для клієнтської та HW частини. Розробив CI скрипти для запуску тестів локально та GitHub actions. Використав бібліотеку covered-unittests для того щоб визначити покриття коду тестами та отримання звіту у XML форматі.

**Посилання**

1. <https://www.arduino.cc/>

2. <https://en.wikipedia.org/wiki/Test_automation>

3. <https://docs.python.org/uk/3/library/tkinter.html>

4. <https://pyserial.readthedocs.io/en/latest/pyserial.html>

5. https://qalight.ua/baza-znaniy/avtomatizovane-testuvannya/