**Міністерство освіти і науки України**

**Національний університет «Запорізька політехніка»**

кафедра програмних засобів

**ЗВІТ**

**З лабораторної роботи №1**

з дисципліни «Агентно-орієнтоване програмування»

Виконав:

студент гр. КНТ-131 Б.В. Телєпов

Прийняли:

Керівник Подковаліхіна О.О.

2025

# РОЗРОБКА АГЕНТІВ У PYTHON ЗА ДОПОМОГОЮ ФРЕЙМВОРКУ SPADE

## Тема та мета роботи.

1. Ознайомитися з основами створення агентів у Python.
2. Навчитися використовувати фреймворк SPADE (Smart Python Agent Development Environment) для побудови агентних систем.
3. Реалізувати агентів, які можуть взаємодіяти між собою через обмін повідомленнями.

## Завдання до роботи.

1. Створити агента, який періодично відправляє повідомлення іншому агенту з випадковим числом.
2. Реалізувати агента, який виконує завдання тільки після отримання певного "ключового" слова в повідомленні.
3. Створити агентну систему, де один агент запитує інформацію у трьох агентів і чекає на їхні відповіді.
4. Реалізувати агента, який зберігає отримані повідомлення у файл.
5. Створити агента, який реагує лише на повідомлення з певною темою (наприклад, “ALERT”).
6. Зробити систему з двома агентами, де один відправляє запит, а інший надсилає відповідь із затримкою.
7. Реалізувати агента, який аналізує текст повідомлення і відповідає залежно від змісту (чат-бот).
8. Створити агента-монітор, який веде лог усіх отриманих повідомлень від різних агентів.
9. Розробити мультиагентну систему для голосування: агенти відправляють свої голоси, а головний агент підраховує результати.
10. Реалізувати агента для обміну числовими даними та обчислення середнього значення

## Виконанння роботи

Створити агента, який періодично відправляє повідомлення іншому агенту з випадковим числом. Результати (рис. 1.1);

import random

import asyncio

from spade.behaviour import CyclicBehaviour

from spade.agent import Agent

from spade.message import Message

# Агент-відправник

class SenderAgent(Agent):

    class SendMessageBehaviour(CyclicBehaviour):

        async def run(self):

            msg = Message(to="studtelepov811@xmpp.jp")  # JID одержувача

            msg.body = str(random.randint(1, 100))

            await self.send(msg)

            print(f"{self.agent.name} sent: {msg.body}")

            await asyncio.sleep(5)  # Відправляємо кожні 5 секунд

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.SendMessageBehaviour())

# Агент-одержувач

class ReceiverAgent(Agent):

    class ReceiveMessageBehaviour(CyclicBehaviour):

        async def run(self):

            msg = await self.receive(timeout=10)  # Очікуємо повідомлення

            if msg:

                print(f"{self.agent.name} received: {msg.body}")

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.ReceiveMessageBehaviour())

async def main():

    sender = SenderAgent("studtelepov811@xmpp.jp", "04082004")

    receiver = ReceiverAgent("studtelepov811@xmpp.jp", "04082004")

    await sender.start()

    await receiver.start()

    await asyncio.sleep(30)  # Час роботи симуляції

    await sender.stop()

    await receiver.stop()

    print("Agents stopped.")

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(main())

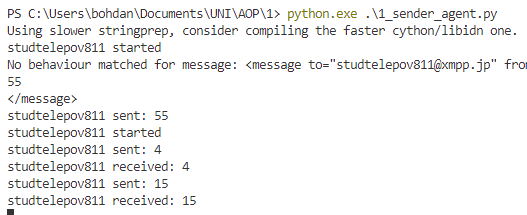


Рисунок 1.1 - Результати роботи

Реалізувати агента, який виконує завдання тільки після отримання певного "ключового" слова в повідомленні. Результати (рис. 1.2);

import random

import asyncio

from spade.behaviour import CyclicBehaviour

from spade.agent import Agent

from spade.message import Message

# Агент-відправник

class SenderAgent(Agent):

    class SendMessageBehaviour(CyclicBehaviour):

        async def run(self):

            msg = Message(to="studtelepov811@xmpp.jp")  # JID одержувача

            msg.body = "This is a key message!" if random.random() > 0.5 else "Just a regular message."

            await self.send(msg)

            print(f"{self.agent.name} sent: {msg.body}")

            await asyncio.sleep(5)

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.SendMessageBehaviour())

# Агент-одержувач (з ключовим словом)

class KeywordAgent(Agent):

    class KeywordBehaviour(CyclicBehaviour):

        async def run(self):

            msg = await self.receive(timeout=10)

            if msg and "key" in msg.body.lower():

                print(f"{self.agent.name} received a key message: {msg.body}")

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.KeywordBehaviour())

async def main():

    sender = SenderAgent("studtelepov811@xmpp.jp", "04082004")

    receiver = KeywordAgent("studtelepov811@xmpp.jp", "04082004")

    await sender.start()

    await receiver.start()

    await asyncio.sleep(30)  # Час роботи симуляції

    await sender.stop()

    await receiver.stop()

    print("Agents stopped.")

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(main())

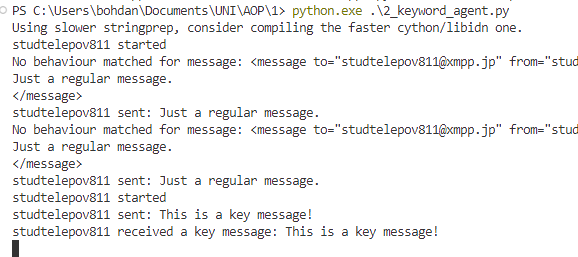


Рисунок 1.2 - Результати роботи

Створити агентну систему, де один агент запитує інформацію у трьох агентів і чекає на їхні відповіді. Результати (рис. 1.3);

import asyncio

from spade.behaviour import CyclicBehaviour, OneShotBehaviour

from spade.agent import Agent

from spade.message import Message

# Агент-запитувач

class QueryAgent(Agent):

    class QueryBehaviour(OneShotBehaviour):

        async def run(self):

            recipients = ["studtelepov811\_r@xmpp.jp"]

            for recipient in recipients:

                msg = Message(to=recipient)

                msg.body = "Request for data"

                await self.send(msg)

            print(f"{self.agent.name} requested information.")

            responses = []

            for \_ in recipients:

                reply = await self.receive(timeout=5)

                if reply:

                    responses.append(reply.body)

            print(f"{self.agent.name} received responses: {responses}")

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.QueryBehaviour())

# Агент-відповідач

class ResponseAgent(Agent):

    class ResponseBehaviour(CyclicBehaviour):  # Використовуємо циклічну поведінку

        async def run(self):

            msg = await self.receive(timeout=10)

            if msg:

                print(f"{self.agent.name} received request from {msg.sender}")

                reply = Message(to=str(msg.sender))

                reply.body = f"Response from {self.agent.name}"

                await self.send(reply)

                print(f"{self.agent.name} replied to {msg.sender}")

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.ResponseBehaviour())

async def main():

    query\_agent = QueryAgent("studtelepov811@xmpp.jp", "04082004")

    agent1 = ResponseAgent("studtelepov811\_r@xmpp.jp", "04082004")

    await agent1.start()

    await query\_agent.start()

    await asyncio.sleep(20)  # Час роботи симуляції

    await query\_agent.stop()

    await agent1.stop()

    print("Agents stopped.")

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(main())

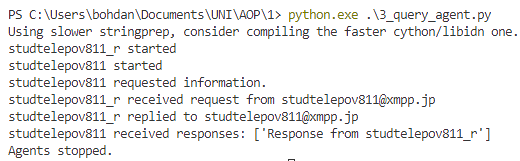


Рисунок 1.3 - Результати роботи

Реалізувати агента, який зберігає отримані повідомлення у файл. Результати (рис. 1.4 - 1.5);

import asyncio

from spade.behaviour import CyclicBehaviour, OneShotBehaviour

from spade.agent import Agent

from spade.message import Message

# Агент-запитувач

class QueryAgent(Agent):

    class QueryBehaviour(OneShotBehaviour):

        async def run(self):

            recipients = ["studtelepov811\_r@xmpp.jp"]

            for recipient in recipients:

                msg = Message(to=recipient)

                msg.body = "Request for data"

                await self.send(msg)

            print(f"{self.agent.name} requested information.")

            responses = []

            for \_ in recipients:

                reply = await self.receive(timeout=5)

                if reply:

                    responses.append(reply.body)

            print(f"{self.agent.name} received responses: {responses}")

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.QueryBehaviour())

# Агент-відповідач

class ResponseAgent(Agent):

    class ResponseBehaviour(CyclicBehaviour):  # Використовуємо циклічну поведінку

        async def run(self):

            msg = await self.receive(timeout=10)

            if msg:

                print(f"{self.agent.name} received request from {msg.sender}")

                reply = Message(to=str(msg.sender))

                reply.body = f"Response from {self.agent.name}"

                await self.send(reply)

                # Логування отриманого повідомлення у файл

                with open("responses.log", "a") as file:

                    file.write(f"Received from {msg.sender}: {msg.body}\n")

                print(f"{self.agent.name} logged message from {msg.sender}")

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.ResponseBehaviour())

async def main():

    query\_agent = QueryAgent("studtelepov811@xmpp.jp", "04082004")

    agent1 = ResponseAgent("studtelepov811\_r@xmpp.jp", "04082004")

    await agent1.start()

    await query\_agent.start()

    await asyncio.sleep(20)  # Час роботи симуляції

    await query\_agent.stop()

    await agent1.stop()

    print("Agents stopped.")

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(main())

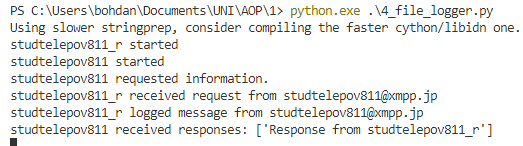


Рисунок 1.4 - Результати роботи

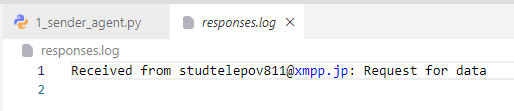


Рисунок 1.5 - Результати роботи

Створити агента, який реагує лише на повідомлення з певною темою (наприклад, “ALERT”). Результати (рис. 1.6);

import asyncio

from spade.behaviour import CyclicBehaviour, OneShotBehaviour

from spade.agent import Agent

from spade.message import Message

# Агент, який реагує лише на повідомлення з темою "ALERT"

class TopicAgent(Agent):

    class TopicBehaviour(CyclicBehaviour):

        async def run(self):

            msg = await self.receive(timeout=10)

            if msg and "ALERT" in msg.body:

                print(f"{self.agent.name} received an alert: {msg.body}")

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.TopicBehaviour())

# Агент-запитувач

class QueryAgent(Agent):

    class QueryBehaviour(OneShotBehaviour):

        async def run(self):

            recipients = ["studtelepov811\_r@xmpp.jp"]

            for recipient in recipients:

                msg = Message(to=recipient)

                msg.body = "ALERT: System failure detected"

                await self.send(msg)

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.QueryBehaviour())

async def main():

    query\_agent = QueryAgent("studtelepov811@xmpp.jp", "04082004")

    topic\_agent = TopicAgent("studtelepov811\_r@xmpp.jp", "04082004")

    # Запуск агентів

    await topic\_agent.start()

    await query\_agent.start()

    await asyncio.sleep(20)  # Час роботи симуляції

    await query\_agent.stop()

    await topic\_agent.stop()

    print("Agents stopped.")

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(main())

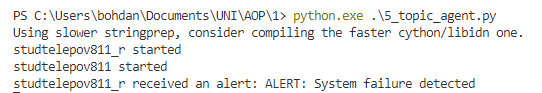


Рисунок 1.6 - Результати роботи

Зробити систему з двома агентами, де один відправляє запит, а інший надсилає відповідь із затримкою. Результати (рис. 1.7);

import asyncio

from spade.behaviour import CyclicBehaviour, OneShotBehaviour

from spade.agent import Agent

from spade.message import Message

# 6 ️. Зробити систему з двома агентами, де один відправляє запит, а інший надсилає відповідь із затримкою.

class DelayedResponseAgent(Agent):

    class RequestHandler(CyclicBehaviour):

        async def run(self):

            msg = await self.receive(timeout=10)

            if msg:

                print(f"{self.agent.name} received a request. Responding after delay...")

                await asyncio.sleep(3)

                reply = msg.make\_reply()

                reply.body = "Delayed Response"

                await self.send(reply)

    async def setup(self):

        self.add\_behaviour(self.RequestHandler())

# Агент-запитувач

class QueryAgent(Agent):

    class QueryBehaviour(OneShotBehaviour):

        async def run(self):

            recipients = ["studtelepov811\_r@xmpp.jp"]

            for recipient in recipients:

                msg = Message(to=recipient)

                msg.body = "ALERT: System failure detected"

                await self.send(msg)

            responses = []

            for \_ in recipients:

                reply = await self.receive(timeout=5)

                if reply:

                    responses.append(reply.body)

            print(f"{self.agent.name} received responses: {responses}")

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.QueryBehaviour())

async def main():

    query\_agent = QueryAgent("studtelepov811@xmpp.jp", "04082004")

    topic\_agent = DelayedResponseAgent("studtelepov811\_r@xmpp.jp", "04082004")

    # Запуск агентів

    await topic\_agent.start()

    await query\_agent.start()

    await asyncio.sleep(20)  # Час роботи симуляції

    await query\_agent.stop()

    await topic\_agent.stop()

    print("Agents stopped.")

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(main())

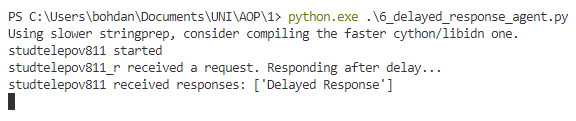


Рисунок 1.7 - Результати роботи

Реалізувати агента, який аналізує текст повідомлення і відповідає залежно від змісту (чат-бот). Результати (рис. 1.8);

import asyncio

from spade.behaviour import CyclicBehaviour, OneShotBehaviour

from spade.agent import Agent

from spade.message import Message

# 7 ️. Реалізувати агента, який аналізує текст повідомлення і відповідає залежно від змісту (чат-бот).

class ChatBotAgent(Agent):

    class ChatBehaviour(CyclicBehaviour):

        async def run(self):

            msg = await self.receive(timeout=10)

            if msg:

                response = "I don't understand."

                if "hello" in msg.body.lower():

                    response = "Hi! How can I help you?"

                elif "bye" in msg.body.lower():

                    response = "Goodbye!"

                reply = msg.make\_reply()

                reply.body = response

                await self.send(reply)

    async def setup(self):

        self.add\_behaviour(self.ChatBehaviour())

# Агент-запитувач

class QueryAgent(Agent):

    class QueryBehaviour(OneShotBehaviour):

        async def run(self):

            recipients = ["studtelepov811\_r@xmpp.jp"]

            for recipient in recipients:

                msg = Message(to=recipient)

                msg.body = "hello"

                await self.send(msg)

                msg.body = "bye"

                await self.send(msg)

            responses = []

            for \_ in recipients:

                reply = await self.receive(timeout=10)

                if reply:

                    responses.append(reply.body)

            print(f"{self.agent.name} received responses: {responses}")

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.QueryBehaviour())

async def main():

    query\_agent = QueryAgent("studtelepov811@xmpp.jp", "04082004")

    topic\_agent = ChatBotAgent("studtelepov811\_r@xmpp.jp", "04082004")

    # Запуск агентів

    await topic\_agent.start()

    await query\_agent.start()

    await asyncio.sleep(20)  # Час роботи симуляції

    await query\_agent.stop()

    await topic\_agent.stop()

    print("Agents stopped.")

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(main())

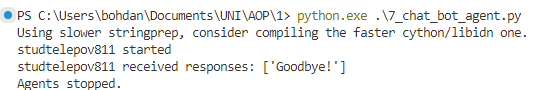


Рисунок 1.8 - Результати роботи

Створити агента-монітор, який веде лог усіх отриманих повідомлень від різних агентів. Результати (рис. 1.9 - 1.10);

import asyncio

from spade.behaviour import CyclicBehaviour, OneShotBehaviour

from spade.agent import Agent

from spade.message import Message

# Агент-монітор, який веде лог усіх отриманих повідомлень

class MonitorAgent(Agent):

    class MonitorBehaviour(CyclicBehaviour):

        async def run(self):

            msg = await self.receive(timeout=10)

            if msg:

                print(f"Monitor received from {msg.sender}: {msg.body}")

                with open("monitor\_log.txt", "a") as file:

                    file.write(f"Received from {msg.sender}: {msg.body}\n")

                print(f"Monitor logged message from {msg.sender}")

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.MonitorBehaviour())

# Агент-запитувач

class QueryAgent(Agent):

    class QueryBehaviour(OneShotBehaviour):

        async def run(self):

            recipients = ["studtelepov811@xmpp.jp"]

            for recipient in recipients:

                msg = Message(to=recipient)

                msg.body = "Hello from QueryAgent"

                await self.send(msg)

            print(f"{self.agent.name} sent a message.")

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.QueryBehaviour())

async def main():

    monitor\_agent = MonitorAgent("studtelepov811@xmpp.jp", "04082004")

    query\_agent = QueryAgent("studtelepov811\_r@xmpp.jp", "04082004")

    # Запуск агентів

    await monitor\_agent.start()

    await query\_agent.start()

    await asyncio.sleep(20)  # Час роботи симуляції

    await monitor\_agent.stop()

    await query\_agent.stop()

    print("Agents stopped.")

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(main())

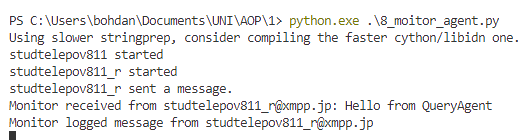


Рисунок 1.9 - Результати роботи

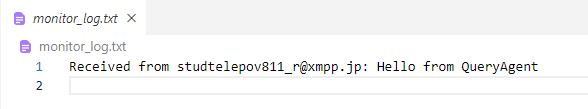


Рисунок 1.10 - Результати роботи

Розробити мультиагентну систему для голосування: агенти відправляють свої голоси, а головний агент підраховує результати. Результати (рис. 1.11 - 1.12);

import random

import asyncio

from spade.behaviour import OneShotBehaviour, CyclicBehaviour

from spade.agent import Agent

from spade.message import Message

# Агент для голосування, який збирає свої голоси

class VotingAgent(Agent):

    class VotingBehaviour(OneShotBehaviour):

        async def run(self):

            votes = [random.choice(["yes", "no"]) for \_ in range(5)]

            print(f"{self.agent.name} collected votes: {votes}")

            result = f"YES: {votes.count('yes')}, NO: {votes.count('no')}"

            print(f"Voting result: {result}")

            msg = Message(to="studtelepov811@xmpp.jp")

            msg.body = result

            await self.send(msg)

    async def setup(self):

        self.add\_behaviour(self.VotingBehaviour())

# Головний агент, який підраховує результати голосування

class MainAgent(Agent):

    class MainBehaviour(CyclicBehaviour):

        async def run(self):

            msg = await self.receive(timeout=10)

            if msg:

                print(f"{self.agent.name} received voting result: {msg.body}")

                with open("voting\_results.log", "a") as file:

                    file.write(f"Result from {msg.sender}: {msg.body}\n")

    async def setup(self):

        print(f"{self.name} started")

        self.add\_behaviour(self.MainBehaviour())

async def main():

    main\_agent = MainAgent("studtelepov811@xmpp.jp", "04082004")

    voting\_agents = [

        VotingAgent("studtelepov811\_r@xmpp.jp", "04082004"),

    ]

    await main\_agent.start()

    for agent in voting\_agents:

        await agent.start()

    await asyncio.sleep(20)  # Час роботи симуляції

    await main\_agent.stop()

    for agent in voting\_agents:

        await agent.stop()

    print("Agents stopped.")

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(main())

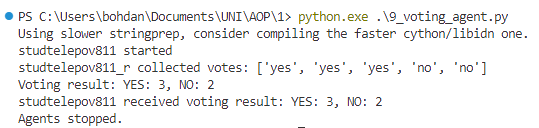


Рисунок 1.11 - Результати роботи

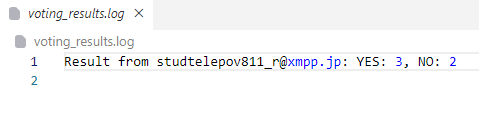


Рисунок 1.12 - Результати роботи

Реалізувати агента для обміну числовими даними та обчислення середнього значення. Результати (рис. 1.13 - 1.14);

import random

import asyncio

from spade.behaviour import CyclicBehaviour

from spade.agent import Agent

from spade.message import Message

# Агент для обміну числовими даними та обчислення середнього значення

class AverageCalculatorAgent(Agent):

    class AverageBehaviour(CyclicBehaviour):

        numbers = []  # Список для зберігання отриманих чисел

        async def run(self):

            msg = await self.receive(timeout=10)  # Чекаємо повідомлення з числовими даними

            if msg:

                self.numbers.append(int(msg.body))

                avg = sum(self.numbers) / len(self.numbers)

                print(f"{self.agent.name} updated average: {avg}")

                with open("average\_log.txt", "a") as file:

                    file.write(f"Received number: {msg.body}, Average: {avg}\n")

    async def setup(self):

        self.add\_behaviour(self.AverageBehaviour())

# Агент-запитувач, який надсилає числові дані

class NumberSenderAgent(Agent):

    class NumberSenderBehaviour(CyclicBehaviour):

        async def run(self):

            numbers\_to\_send = [random.randint(1, 100) for \_ in range(5)]

            for num in numbers\_to\_send:

                msg = Message(to="studtelepov811@xmpp.jp")

                msg.body = str(num)

                await self.send(msg)

                print(f"{self.agent.name} sent number: {num}")

            self.kill()

    async def setup(self):

        self.add\_behaviour(self.NumberSenderBehaviour())

async def main():

    average\_calculator = AverageCalculatorAgent("studtelepov811@xmpp.jp", "04082004")

    number\_sender = NumberSenderAgent("studtelepov811\_r@xmpp.jp", "04082004")

    # Запуск агентів

    await average\_calculator.start()

    await number\_sender.start()

    await asyncio.sleep(20)  # Час роботи симуляції

    await average\_calculator.stop()

    await number\_sender.stop()

    print("Agents stopped.")

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(main())

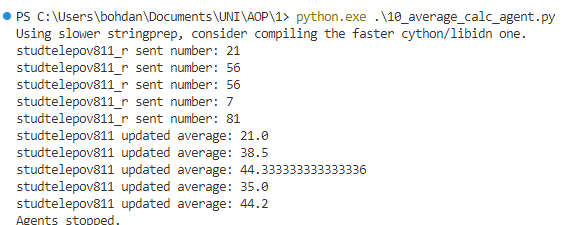


Рисунок 1.13 - Результати роботи

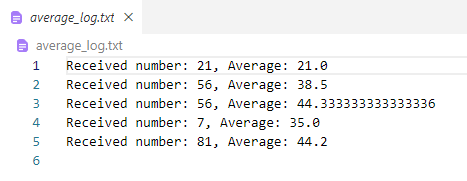


Рисунок 1.14 - Результати роботи

## Відповіді на запитання

1. ***Що таке SPADE і для чого він використовується?***

Фреймворк для Python, використовується для моделювання агентних систем, розподілених обчислень та автоматизації.

1. ***Які основні компоненти агента у SPADE?***

Agent – основний клас для створення агентів.

Behaviour – поведінка агента, яка визначає його дії.

Message – об'єкт для передачі інформації між агентами.

JID (Jabber ID) – унікальний ідентифікатор агента в XMPP-мережі.

1. ***Як агенти взаємодіють між собою у SPADE?***

Через відправку повідомлень у XMPP-мережі.

1. ***Що таке поведінка агента (Behaviour)?***

Це клас, що визначає, як агент поводиться (реагує на події).

1. ***У чому відмінність між CyclicBehaviour і OneShotBehaviour?***

CyclicBehaviour - використовується для постійного очікування повідомлень або моніторингу стану.

OneShotBehaviour - використовується для одноразових операцій, наприклад, ініціалізації або відправки даних.

1. ***Як налаштувати обліковий запис XMPP для агента?***

- Зареєструвати обліковий запис на XMPP-сервері (наприклад, xmpp.jp або jabber.at).

- Отримати JID та пароль, наприклад, my\_agent@xmpp.jp.

agent = MyAgent("my\_agent@xmpp.jp", "password")

agent.start()

1. ***Що таке асинхронне програмування і чому воно важливе у SPADE?***

Асинхронне програмування дозволяє виконувати кілька завдань одночасно без блокування основного потоку.

У SPADE є важливим, бо:

- Дозволяє агентам чекати повідомлення, не блокуючи весь процес.

- Підвищує ефективність комунікації між агентами.

- Використовує async/await, що дозволяє запускати агентів у фоновому режимі.

1. ***Як агент може зберігати отримані дані?***

Через будь-які варіанти зберігання даних в залежнсоті від прописаної логіки у тілі агента. (включаючи базовий self.data)

1. ***Які протоколи використовуються для комунікації агентів у SPADE?***

XMPP (Extensible Messaging and Presence Protocol) – основний протокол для обміну повідомленнями між агентами.

FIPA-ACL (Foundation for Intelligent Physical Agents - Agent Communication Language) – стандартний агентний протокол для координації дій.

XML (Extensible Markup Language) – використовується в XMPP для структурування повідомлень.

TLS (Transport Layer Security) – забезпечує безпеку обміну повідомленнями.

10. ***Як забезпечити безпеку обміну повідомленнями між агентами?***

Шифрувати повідомлення.