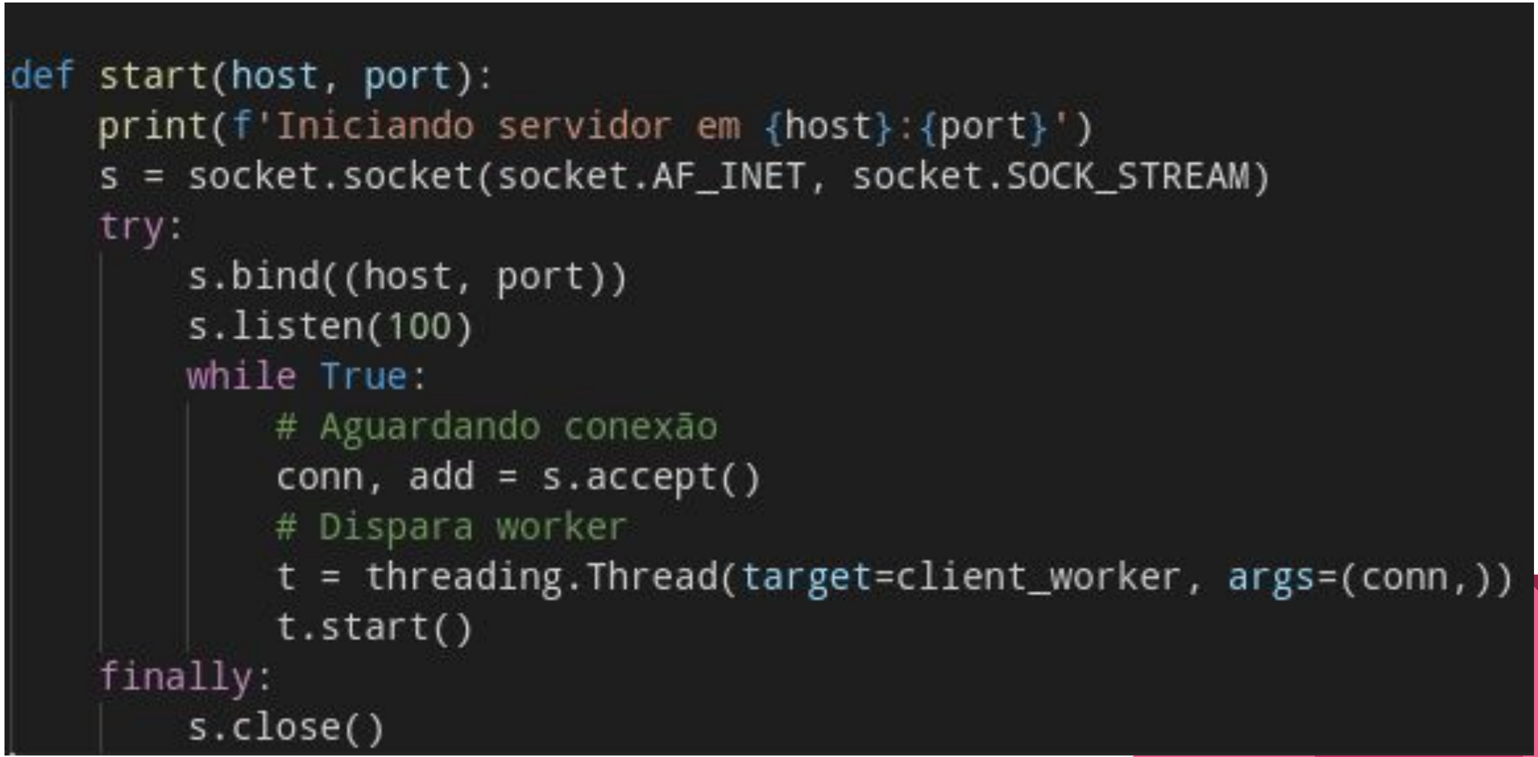
Exercício 1 - Middleware

socket Python

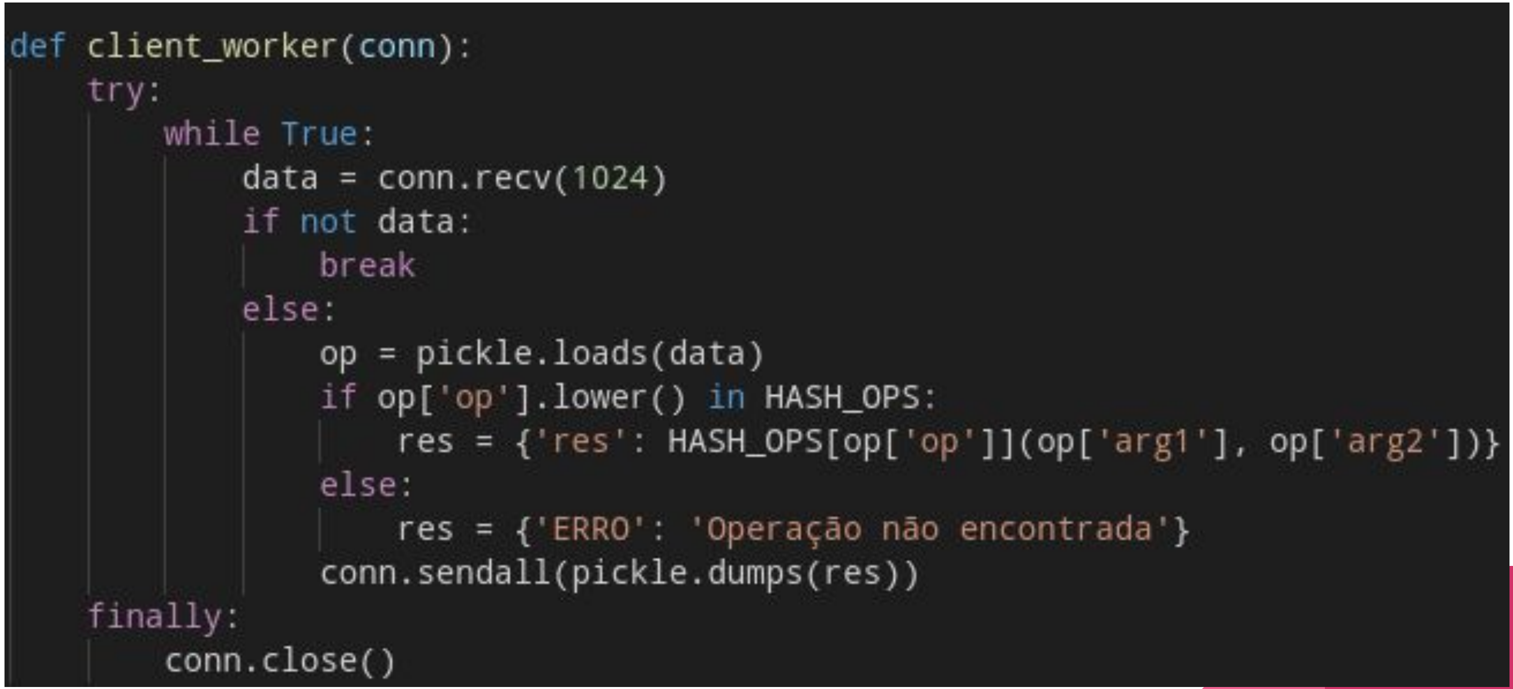
Adilson Angelo (aasj2)

Servidor TCP



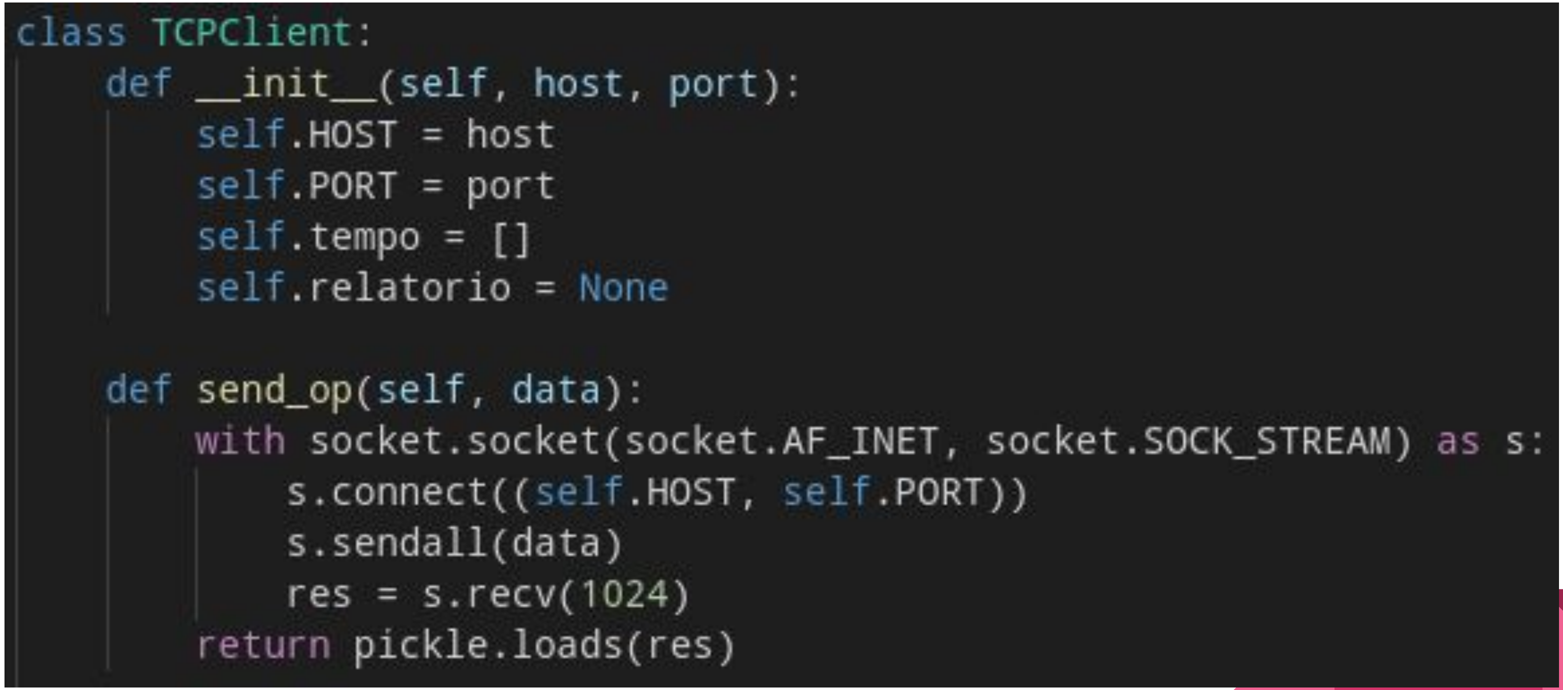
```
def start(host, port):
    print(f'Iniciando servidor em {host}:{port}')
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        s.bind((host, port))
        s.listen(100)
        while True:
            # Aguardando conexão
            conn, add = s.accept()
            # Dispara worker
            t = threading.Thread(target=client_worker, args=(conn,))
            t.start()
    finally:
        s.close()
```

Servidor TCP



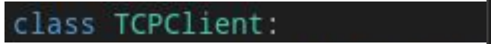
```python
def client_worker(conn):
    try:
        while True:
            data = conn.recv(1024)
            if not data:
                break
            else:
                op = pickle.loads(data)
                if op['op'].lower() in HASH_OPS:
                    res = {'res': HASH_OPS[op['op']](op['arg1'], op['arg2'])}
                else:
                    res = {'ERRO': 'Operação não encontrada'}
                conn.sendall(pickle.dumps(res))
    finally:
        conn.close()
```

Cliente TCP

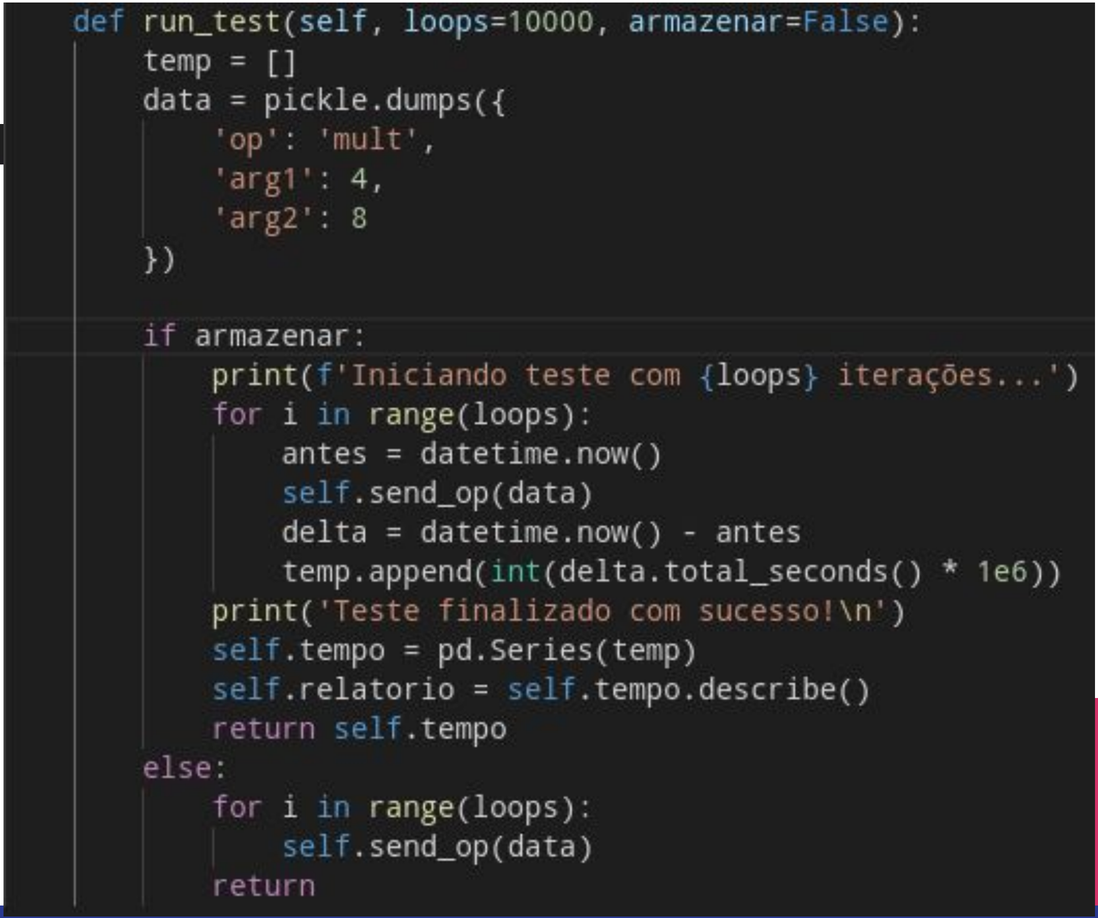


```python
class TCPClient:
    def __init__(self, host, port):
        self.HOST = host
        self.PORT = port
        self.tempo = []
        self.relatorio = None

    def send_op(self, data):
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.connect((self.HOST, self.PORT))
            s.sendall(data)
            res = s.recv(1024)
        return pickle.loads(res)
```

Cliente TCP

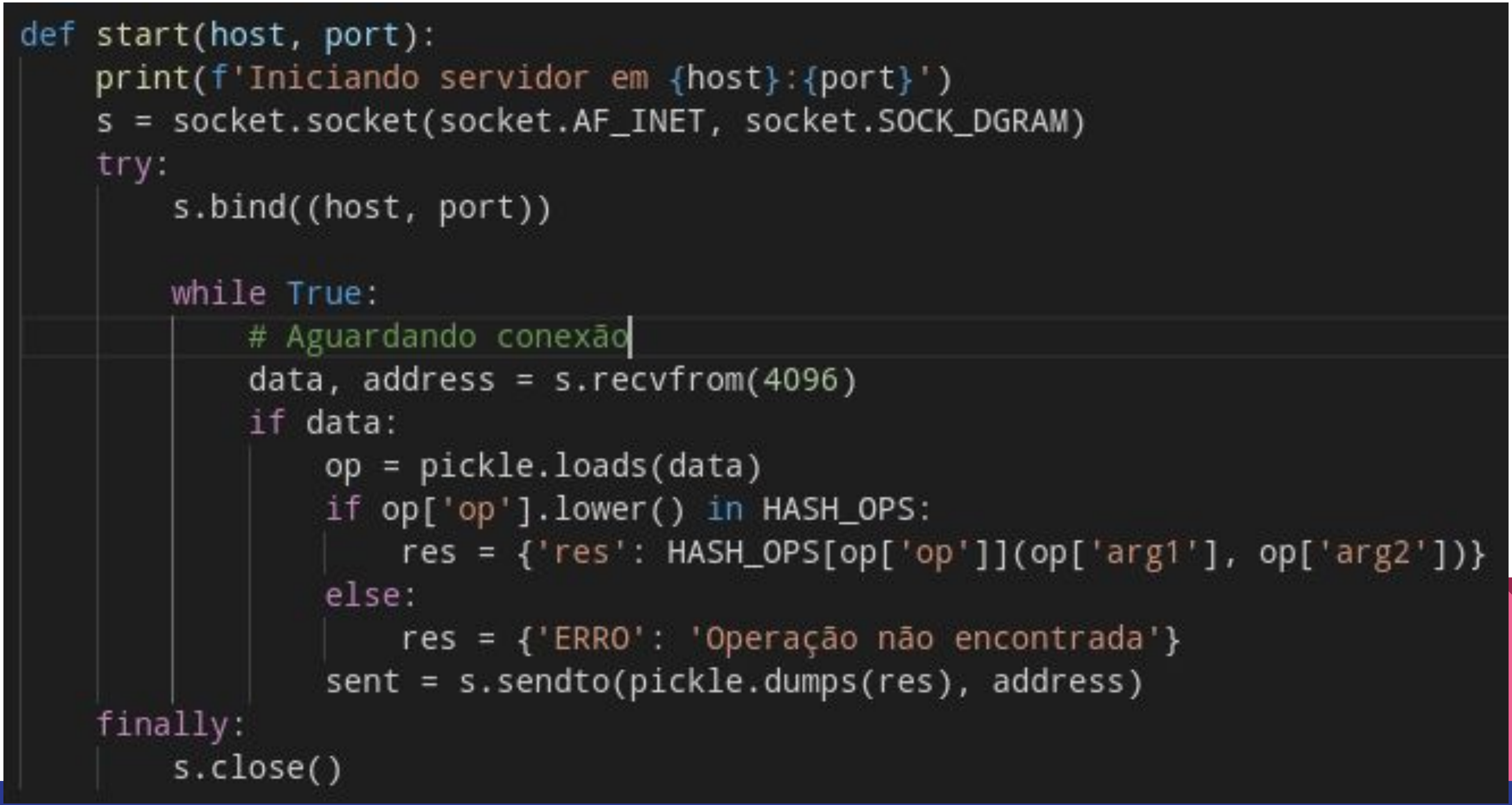


```python
class TCPClient:
```



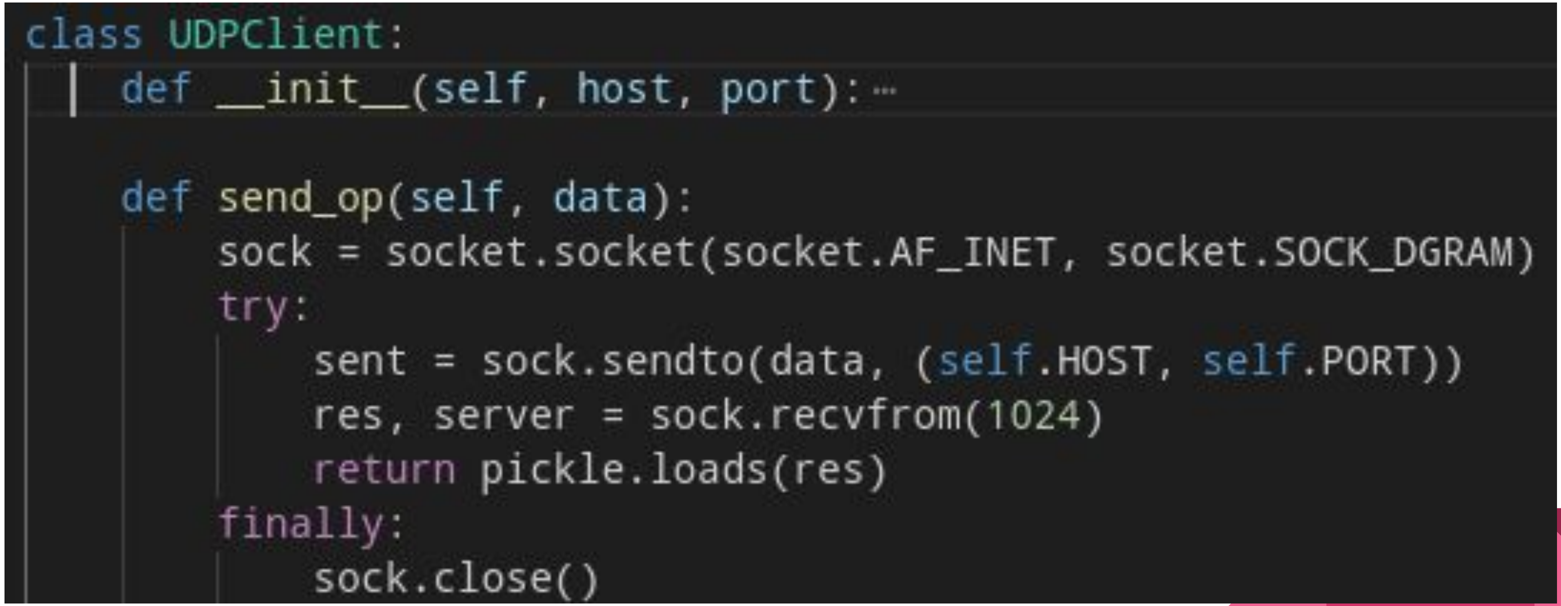
```python
    def run_test(self, loops=10000, armazenar=False):
        temp = []
        data = pickle.dumps({
            'op': 'mult',
            'arg1': 4,
            'arg2': 8
        })

        if armazenar:
            print(f'Iniciando teste com {loops} iterações...')
            for i in range(loops):
                antes = datetime.now()
                self.send_op(data)
                delta = datetime.now() - antes
                temp.append(int(delta.total_seconds() * 1e6))
            print('Teste finalizado com sucesso!\n')
            self.tempo = pd.Series(temp)
            self.relatorio = self.tempo.describe()
            return self.tempo
        else:
            for i in range(loops):
                self.send_op(data)
            return
```

Servidor UDP



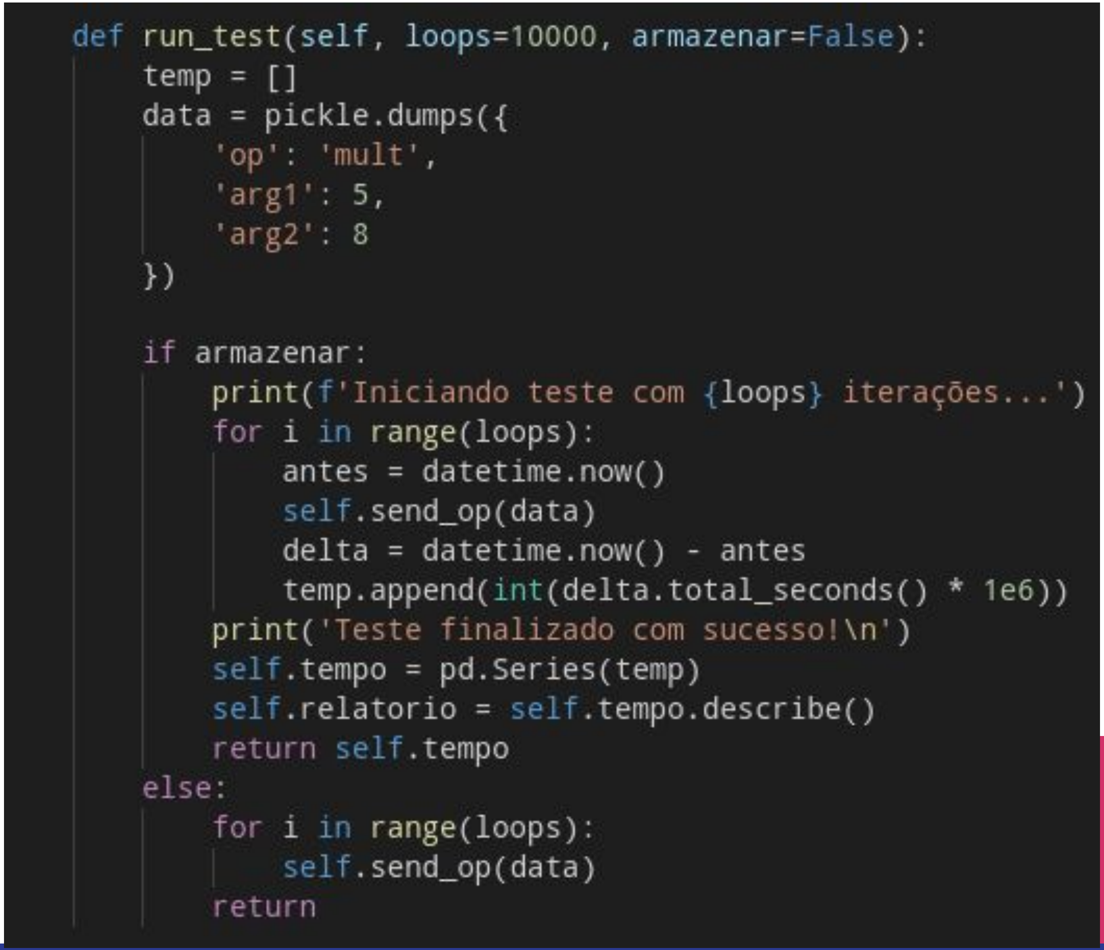
```python
def start(host, port):
    print(f'Iniciando servidor em {host}:{port}')
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        s.bind((host, port))

        while True:
            # Aguardando conexão
            data, address = s.recvfrom(4096)
            if data:
                op = pickle.loads(data)
                if op['op'].lower() in HASH_OPS:
                    res = {'res': HASH_OPS[op['op']](op['arg1'], op['arg2'])}
                else:
                    res = {'ERRO': 'Operação não encontrada'}
                sent = s.sendto(pickle.dumps(res), address)
    finally:
        s.close()
```

Cliente UDP



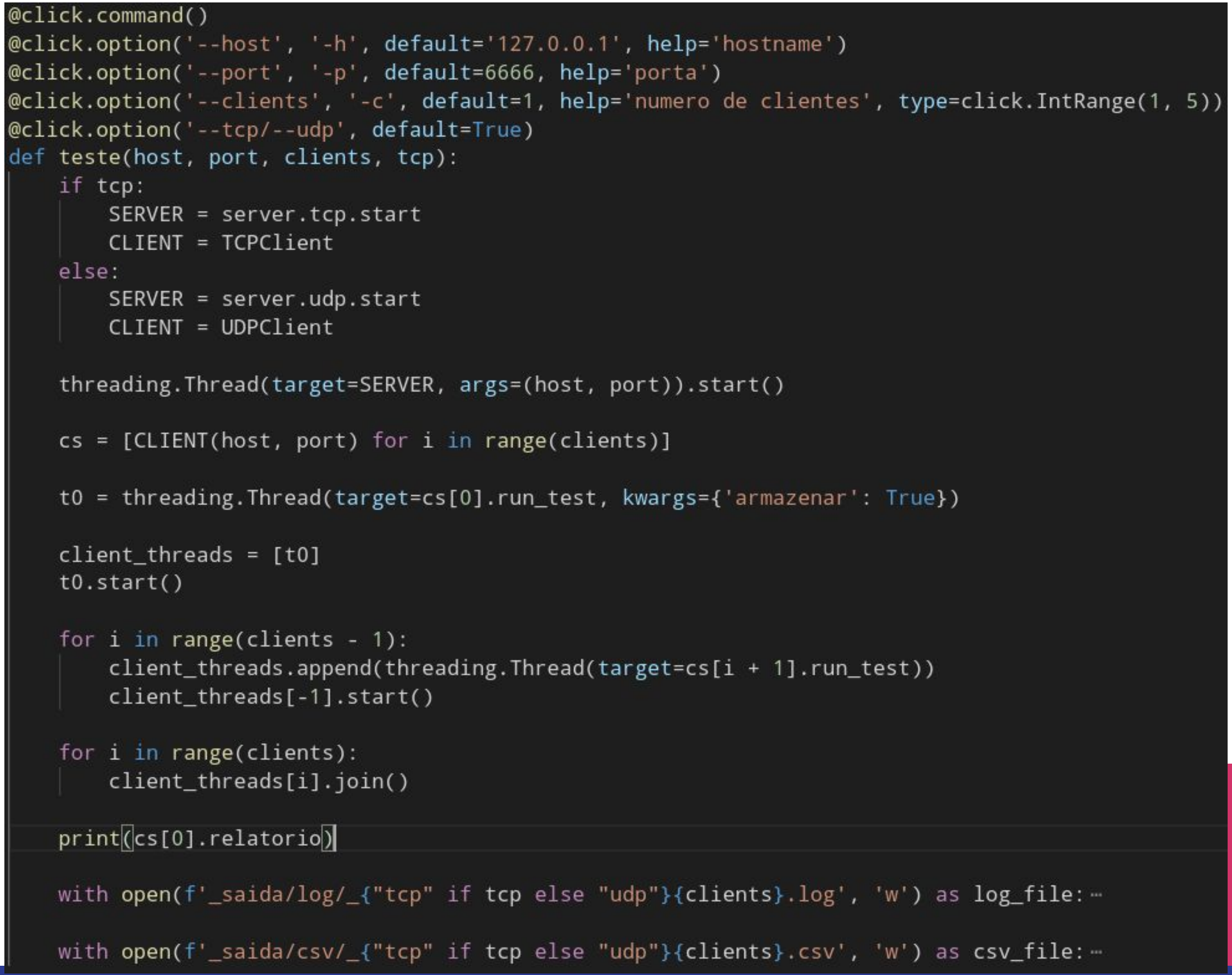
```
class UDPClient:
    def __init__(self, host, port):…

    def send_op(self, data):
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        try:
            sent = sock.sendto(data, (self.HOST, self.PORT))
            res, server = sock.recvfrom(1024)
            return pickle.loads(res)
        finally:
            sock.close()
```

Cliente UDP



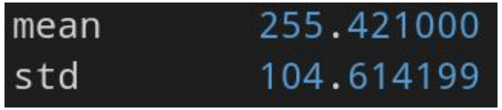
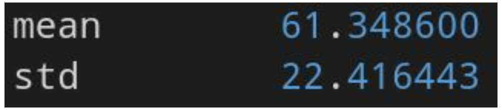
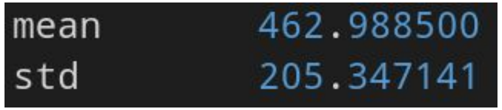
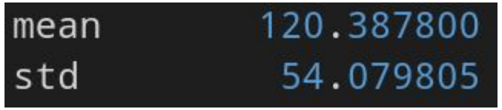
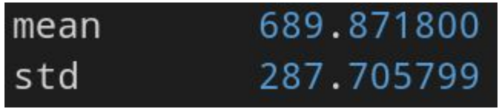
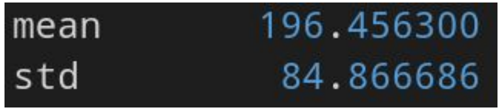
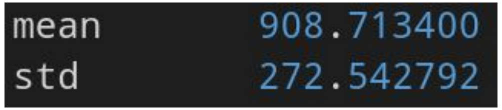
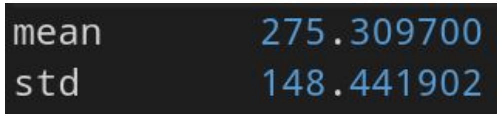
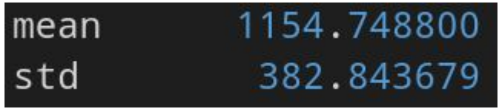
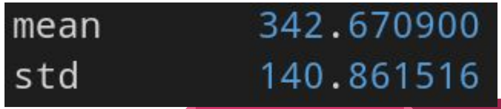
```
def run_test(self, loops=10000, armazenar=False):
    temp = []
    data = pickle.dumps({
        'op': 'mult',
        'arg1': 5,
        'arg2': 8
    })

    if armazenar:
        print(f'Iniciando teste com {loops} iterações...')
        for i in range(loops):
            antes = datetime.now()
            self.send_op(data)
            delta = datetime.now() - antes
            temp.append(int(delta.total_seconds() * 1e6))
        print('Teste finalizado com sucesso!\n')
        self.tempo = pd.Series(temp)
        self.relatorio = self.tempo.describe()
        return self.tempo
    else:
        for i in range(loops):
            self.send_op(data)
        return
```

Teste

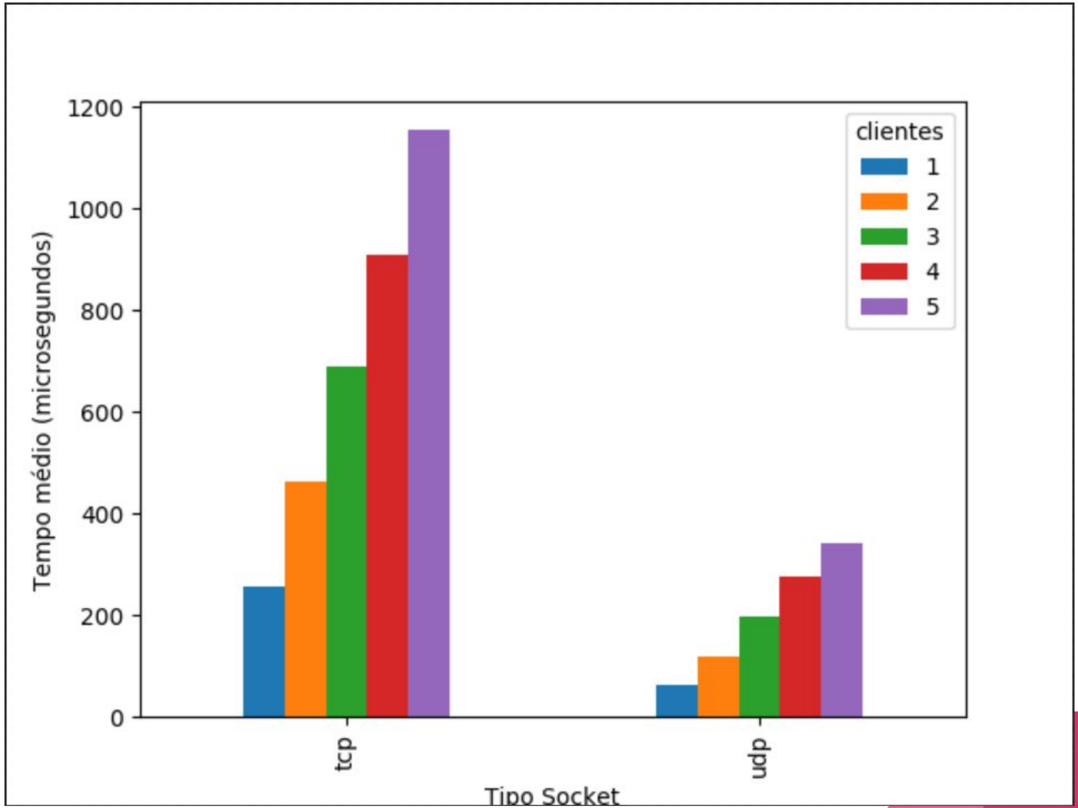


```python
@click.command()
@click.option('--host', '-h', default='127.0.0.1', help='hostname')
@click.option('--port', '-p', default=6666, help='porta')
@click.option('--clients', '-c', default=1, help='numero de clientes', type=click.IntRange(1, 5))
@click.option('--tcp/--udp', default=True)
def teste(host, port, clients, tcp):
    if tcp:
        SERVER = server.tcp.start
        CLIENT = TCPClient
    else:
        SERVER = server.udp.start
        CLIENT = UDPClient

    threading.Thread(target=SERVER, args=(host, port)).start()

    cs = [CLIENT(host, port) for i in range(clients)]

    t0 = threading.Thread(target=cs[0].run_test, kwargs={'armazenar': True})

    client_threads = [t0]
    t0.start()

    for i in range(clients - 1):
        client_threads.append(threading.Thread(target=cs[i + 1].run_test))
        client_threads[-1].start()

    for i in range(clients):
        client_threads[i].join()

    print(cs[0].relatorio)

    with open(f'_saida/log/_{"tcp" if tcp else "udp"}{clients}.log', 'w') as log_file:⋯

    with open(f'_saida/csv/_{"tcp" if tcp else "udp"}{clients}.csv', 'w') as csv_file:⋯
```

Resultados

Clientes	TCP	UDP
1	 `mean 255.421000` `std 104.614199`	 `mean 61.348600` `std 22.416443`
2	 `mean 462.988500` `std 205.347141`	 `mean 120.387800` `std 54.079805`
3	 `mean 689.871800` `std 287.705799`	 `mean 196.456300` `std 84.866686`
4	 `mean 908.713400` `std 272.542792`	 `mean 275.309700` `std 148.441902`
5	 `mean 1154.748800` `std 382.843679`	 `mean 342.670900` `std 140.861516`

Resultados



clientes
1
2
3
4
5
1200
1000
800
600
400
200
0
Tempo médio (microsegundos)
tcp
udp
Tipo Socket