

```

1  #pagella client per pagellaServerMulti_V01.py versione 1.0 multithread
2  #nome del file : pagellaClientMulti_V01.py
3  import socket
4  import sys
5  import random
6  import os
7  import time
8  import threading
9  import multiprocessing
10 import json
11 import pprint
12
13 SERVER_ADDRESS = '127.0.0.1'
14 SERVER_PORT = 22001
15 NUM_WORKERS=1
16
17 def genera_richieste1(num,address,port):
18     #start_time_thread= time.time()
19     try:
20         s=socket.socket()
21         s.connect((address,port))
22         print(f"\n{threading.current_thread().name} {num+1}) Connessione al server: {address}:{port}")
23     except:
24         print(f"{threading.current_thread().name} Qualcosa è andato storto, sto uscendo... \n")
25         sys.exit()
26
27     #1. Generazione casuale:
28     # di uno studente(valori ammessi: 5 cognomi a caso scelti da una lista)
29     # di una materia(valori ammessi: Matematica, Italiano, inglese, Storia e Geografia)
30     # di un voto (valori ammessi 1 ..10)
31     # delle assenze (valori ammessi 1..5)
32     studenti=['Studente0','Studente1','Studente2','Studente3','Studente4']
33     materie=['Matematica','Italiano','inglese','Storia e Geografia']
34     voto=random.randint(1,10)
35     assenze=random.randint(1,5)
36     materia=materie[random.randint(0,3)]
37     studente=studenti[random.randint(0,4)]
38
39     #2. comporre il messaggio, inviarlo come json e ricevere il risultato
40     messaggio={'studente':studente,
41               'materia':materia,
42               'voto':voto,
43               'assenze':assenze}
44     print(f"Dati inviati al server {messaggio}")
45     messaggio=json.dumps(messaggio)
46     s.sendall(messaggio.encode("UTF-8"))
47     data=s.recv(1024)
48     data=data.decode()
49     data=json.loads(data)
50     print(f"Dati ricevuti dal server {data}")
51
52     if not data:
53         print(f"{threading.current_thread().name}: Server non risponde. Exit")
54     else:
55         # completare:
56         #1. recuperare studente, materia, voto e assenze
57         studente=data['studente']
58         materia=data['materia']
59         print(f"{threading.current_thread().name}: La valutazione di {data['studente']} in {data['materia']} è {data['valutazione']} ")
60     s.close()
61     #end_time_thread=time.time()
62     #print(f"{threading.current_thread().name} tempo di esecuzione time=", end_time_thread-start_time_thread)

```

```

64 def genera_richieste2(num,address,port):
65     #start_time_thread= time.time()
66     try:
67         s=socket.socket()
68         s.connect((address,port))
69         print(f"\n{threading.current_thread().name} {num+1}) Connessione al server: {address}:{port}")
70     except:
71         print(f"{threading.current_thread().name} Qualcosa è andato storto, sto uscendo... \n")
72         sys.exit()
73
74     #1. Generazione casuale di uno studente(valori ammessi: 5 cognomi a caso scelti da una lista)
75     # Per ognuna delle materie ammesse: Matematica, Italiano, inglese, Storia e Geografia)
76     # generazione di un voto (valori ammessi 1..10)
77     # e delle assenze (valori ammessi 1..5)
78     # esempio: pagella={'studente': 'Studente2', 'pagella': [('Matematica', 1, 1), ('Italiano', 3, 3), ('Inglese', 5, 4), ('Storia e Geografia', 1, 1)]}
79
80     studenti=['Studente0','Studente1','Studente2','Studente3','Studente4']
81     materie=['Matematica','Italiano','inglese','Storia e Geografia']
82     studente=studenti[random.randint(0,4)]
83     pagella=[]
84     for m in materie:
85         voto=random.randint(1,10)
86         assenze=random.randint(1,5)
87         pagella.append((m,voto,assenze))
88
89     #2. comporre il messaggio, inviarlo come json e ricevere il risultato
90     messaggio={'studente':studente,
91               'pagella':pagella}
92     print(f"Dati inviati al server {messaggio}")
93     messaggio=json.dumps(messaggio)
94     s.sendall(messaggio.encode("UTF-8"))
95     data=s.recv(1024)
96     data=data.decode()
97     data=json.loads(data)
98     print(f"Dati ricevuti dal server {data}")
99
100     if not data:
101         print(f"{threading.current_thread().name}: Server non risponde. Exit")
102     else:
103         # completare:
104         #1. recuperare studente, media voti e totale assenze
105
106         print(f"{threading.current_thread().name}: Lo studente {data['studente']} ha una media di: {data['media']:.2f} e un totale di assenze : {data['assenze']} ")
107     s.close()
108     #end_time_thread=time.time()
109     #print(f"{threading.current_thread().name} tempo di esecuzione time=", end_time_thread-start_time_thread)

```

```

111 def genera_richieste3(num,address,port):
112     #start_time_thread= time.time()
113     try:
114         s=socket.socket()
115         s.connect((address,port))
116         print(f"\n{threading.current_thread().name} {num+1}) Connessione al server: {address}:{port}")
117     except:
118         print(f"{threading.current_thread().name} Qualcosa è andato storto, sto uscendo... \n")
119         sys.exit()
120
121     # 1. Per ognuno degli studenti ammessi: 5 cognomi a caso scelti da una lista
122     # Per ognuna delle materie ammesse: Matematica, Italiano, inglese, Storia e Geografia)
123     # generazione di un voto (valori ammessi 1..10)
124     # e delle assenze (valori ammessi 1..5)
125     # esempio: tabellone={"Cognome1":[{"Matematica",8,1), ("Italiano",6,1), ("Inglese",9,3), ("Storia",8,2), ("Geografia",8,1)],
126     #                      "Cognome2":[{"Matematica",7,2), ("Italiano",5,3), ("Inglese",4,12), ("Storia",5,2), ("Geografia",4,1)],
127     #                      ....}
128     #2. comporre il messaggio, inviarlo come json
129
130     studenti=['Studente0','Studente1','Studente2','Studente3','Studente4']
131     materie=['Matematica','Italiano','inglese','Storia e Geografia']
132     tabellone={}
133     for stud in studenti:
134         pagella=[]
135         for m in materie:
136             voto=random.randint(1,10)
137             assenze=random.randint(1,5)
138             pagella.append((m,voto,assenze))
139         tabellone[stud]=pagella
140     #2. comporre il messaggio, inviarlo come json e ricevere il risultato
141     #messaggio={'studente':studente,
142               #'pagella':pagella}
143     print("Dati inviati al server")
144     pp=pprint.PrettyPrinter(indent=4)
145     pp.pprint(tabellone)
146     tabellone=json.dumps(tabellone)
147     s.sendall(tabellone.encode("UTF-8"))
148     data=s.recv(1024)
149     data=data.decode()
150     data=json.loads(data)
151     print("Dati ricevuti dal server")
152     pp.pprint(data)
153
154     if not data:
155         print(f"{threading.current_thread().name}: Server non risponde. Exit")
156     else:
157         # completare:
158         #1. recuperare il tabellone e stampare i dettagli di ogni singolo studente
159         for elemento in data:
160             print(f"{threading.current_thread().name}: Lo studente {elemento['studente']} ha una media di: {elemento['media']:.2f} e un totale di assenze : {elemento['a
161         s.close()
162         #end_time_thread=time.time()
163         #print(f"{threading.current_thread().name} tempo di esecuzione time=", end_time_thread-start_time_thread)

```

```

166 if __name__ == '__main__':
167     start_time=time.time()
168     # PUNTO A) ciclo per chiamare NUM_WORKERS volte la funzione genera richieste (1,2,3)
169     # alla quale passo i parametri (num,SERVER_ADDRESS, SERVER_PORT)
170     for num in range (0,NUM_WORKERS):
171         genera_richieste1(num,SERVER_ADDRESS, SERVER_PORT)
172         #genera_richieste2(num,SERVER_ADDRESS, SERVER_PORT)
173         #genera_richieste3(num,SERVER_ADDRESS, SERVER_PORT)
174     end_time=time.time()
175     print("Total SERIAL time=", end_time - start_time)
176
177     start_time=time.time()
178     threads=[]
179     # PUNTO B) ciclo per chiamare NUM_WORKERS volte la funzione genera richieste (1,2,3)
180     # tramite l'avvio di un thread al quale passo i parametri args=(num,SERVER_ADDRESS, SERVER_PORT,)
181     # avviare tutti i thread e attenderne la fine
182     for num in range(NUM_WORKERS):
183         threads.append(threading.Thread(target=genera_richieste1, args=(num,SERVER_ADDRESS, SERVER_PORT,)))
184         #threads.append(threading.Thread(target=genera_richieste2, args=(num,SERVER_ADDRESS, SERVER_PORT,)))
185         #threads.append(threading.Thread(target=genera_richieste3, args=(num,SERVER_ADDRESS, SERVER_PORT,)))
186     [thread.start() for thread in threads]
187     [thread.join() for thread in threads]
188     end_time=time.time()
189     print("Total THREADS time= ", end_time - start_time)
190
191     start_time=time.time()
192     process=[]
193     # PUNTO C) ciclo per chiamare NUM_WORKERS volte la funzione genera richieste (1,2,3)
194     # tramite l'avvio di un processo al quale passo i parametri args=(num,SERVER_ADDRESS, SERVER_PORT,)
195     # avviare tutti i processi e attenderne la fine
196     for num in range(NUM_WORKERS):
197         process.append(multiprocessing.Process(target=genera_richieste1, args=(num,SERVER_ADDRESS, SERVER_PORT,)))
198         #process.append(multiprocessing.Process(target=genera_richieste2, args=(num,SERVER_ADDRESS, SERVER_PORT,)))
199         #process.append(multiprocessing.Process(target=genera_richieste3, args=(num,SERVER_ADDRESS, SERVER_PORT,)))
200     [process.start() for process in process]
201     [process.join() for process in process]
202     end_time=time.time()
203     print("Total PROCESS time= ", end_time - start_time)

```