PYTHON SNIPPETS

Contents

Digital Clock using python

tkinter comes with the Python3 standard distribution.

TK() object represents the main window of the application. The Title can be customized using the .title('string') method.

Label() objects are widgets that can display text and images. The style of the label object can be set using its constructor .They can be updated periodically using the .after() method. They can be set using the .config() method.

The strftime() method from the time module can be used to create date and time strings.

Python messenger using LAN

We create a server and a client that can communicate over LAN.

The server waits for a message from the client and prints it. On receiving the message ‘q’ it quits. It takes input from the user and sends it to the client.

The client sends messages to the server, with input from the user, and prints the received message.

Both the client and the server run as sets of 2 threads each, with the server receiving first and the client sending a message.

The code has been improved with context managers to manage the threads better.

Logging has been added.

server.py:

```
"""
A server script.The server waits for a message from the client and lg.infos
it. On receiving the message ‘q’ it quits. It takes input from the user and
sends it to the client.

Attributions:

S: ding.wav by tim.kahn ( paypal.me/pools/c/83tOZtfkXU) -- https://freesound.org/s/91926/ -- License: Attribution
S: buttonchime02up.wav by JustinBW -- https://freesound.org/s/80921/ -- License: Attribution
"""
from socket import *# pylint: disable=unused-wildcard-import
import threading
import sys
from playsound import playsound
import logging
import logging.config
from json import load as jload
import concurrent.futures

# Configure logger lg with config for appLogger from config.json["logging"]
with open("config.json", "r") as f:
    config = jload(f)
    logging.config.dictConfig(config["logging"])
lg = logging.getLogger("appLogger")
# lg.debug("This is a debug message")

# server code
flag = False

def receive_from_client(conn):
    global flag
    try:
        while True:
            if flag == True:
                break
            message = conn.recv(1024).decode()

            if message == 'q':
                conn.send('q'.encode())
                lg.info('Closing connection')
                conn.close()

                flag = True
                break
            lg.info(f'Client: {message}')
            playsound('80921__justinbw__buttonchime02up.wav')

    except:
        conn.close()

def send_to_client(conn):
    global flag
    try:
        while True:
            if flag == True:
                break
            send_message = input('')

            if send_message == 'q':
```

```
                conn.send('q'.encode())
                lg.info('Closing connection')
                conn.close()

                flag = True
                break
            conn.send(send_message.encode())
    except:
        conn.close()


def main():
    # threads = []

    global flag

    host = '192.168.1.6'

    server_port = 6789
    server_socket = socket(AF_INET, SOCK_STREAM)
    server_socket.bind((host,server_port))

    server_socket.listen(1)

    lg.info('The server is ready to connect to a chat client')

    connection_socket, _ = server_socket.accept()

    lg.info('The server is connected to a chat client')

    # t_rcv = threading.Thread(target=receive_from_client, args=(connection_socket,))
    # t_snd = threading.Thread(target=send_to_client, args=(connection_socket,))

    # threads.append(t_rcv)
    # threads.append(t_snd)

    # t_rcv.start()
    # t_snd.start()

    # t_rcv.join()
    # t_snd.join()

    with concurrent.futures.ThreadPoolExecutor(max_workers=2) as executor:
        recieve_future = executor.submit(receive_from_client, connection_socket)
        send_future = executor.submit(send_to_client, connection_socket)

    try:
        lg.info(send_future.result())
    except Exception as exc:
        lg.error(exc)

    try:
        lg.info(recieve_future.result())
    except Exception as exc:
        lg.error(exc)

    lg.info('Server: exiting')

    server_socket.close()

    sys.exit()

if __name__ == "__main__":
    main()
```

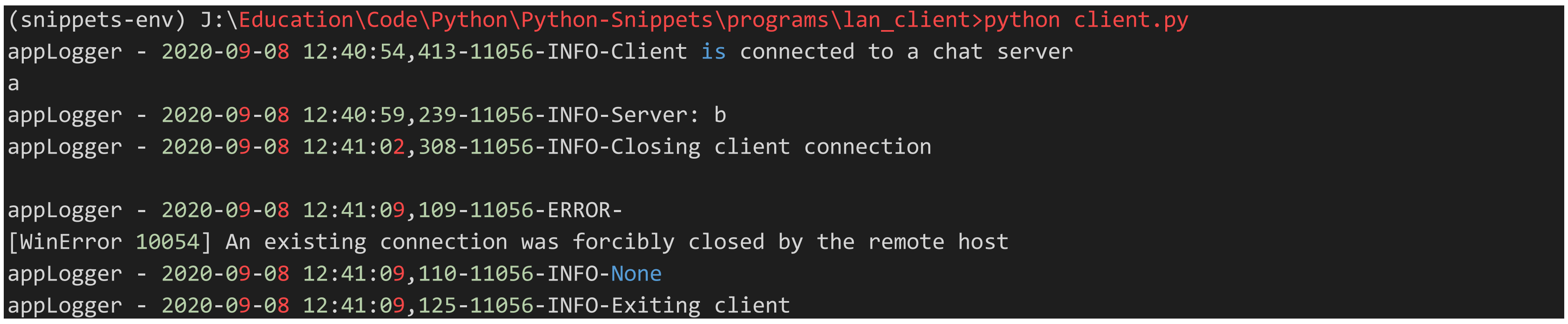
```
(snippets-env) J:\Education\Code\Python\Python-Snippets\programs\lan_client>python server.py
appLogger - 2020-09-08 12:40:50,689-11032-INFO-The server is ready to connect to a chat client
appLogger - 2020-09-08 12:40:54,413-11032-INFO-The server is connected to a chat client
appLogger - 2020-09-08 12:40:56,147-11032-INFO-Client: a
b
q
appLogger - 2020-09-08 12:41:02,308-11032-INFO-Closing connection
```

```
appLogger - 2020-09-08 12:41:02,312-11032-INFO-None
appLogger - 2020-09-08 12:41:02,330-11032-INFO-None
appLogger - 2020-09-08 12:41:02,332-11032-INFO-Server: exiting
```

client.py:

```python
"""
The client sends messages to the server, with input from the user, and prints
the received message.

Attributions:

S: ding.wav by tim.kahn ( paypal.me/pools/c/83tOZtfkXU) -- https://freesound.org/s/91926/ -- License: Attribution
S: buttonchime02up.wav by JustinBW -- https://freesound.org/s/80921/ -- License: Attribution
"""
from socket import *# pylint: disable=unused-wildcard-import
import threading
import sys
from playsound import playsound
import logging
import logging.config
from json import load as jload
import concurrent.futures



# Configure logger lg with config for appLogger from config.json["logging"]
with open("config_copy.json", "r") as f:
    config = jload(f)
    logging.config.dictConfig(config["logging"])
lg = logging.getLogger("appLogger")
# lg.debug("This is a debug message")

# server code
FLAG = False

def send_to_server(clsock):
    global FLAG
    while True:
        if FLAG == True:
            break
        send_message = input('')
        clsock.sendall(send_message.encode())

def recieve_from_server(clsock):
    global FLAG
    while True:
        data = clsock.recv(1024).decode()
        if data == 'q':
            lg.info('Closing client connection')
            FLAG = True
            break
        lg.info(f'Server: {data}')
        playsound('91926__tim-kahn__ding.wav')

def main():
    # threads = []

    host = '192.168.1.6'
    port = 6789

    client_socket = socket(AF_INET, SOCK_STREAM)

    client_socket.connect((host, port))
    lg.info('Client is connected to a chat server')

    # t_send = threading.Thread(target=send_to_server, args=(client_socket,))
    # t_recv = threading.Thread(target=recieve_from_server, args=(client_socket,))

    # threads.append(t_send)
    # threads.append(t_recv)

    # t_send.start()
```

```
    # t_recv.start()

    # t_send.join()
    # t_recv.join()

    with concurrent.futures.ThreadPoolExecutor(max_workers=2) as executor:
        send_future = executor.submit(send_to_server, client_socket)
        recieve_future = executor.submit(recieve_from_server, client_socket)

    try:
        lg.info(send_future.result())
    except Exception as exc:
        lg.error(exc)

    try:
        lg.info(recieve_future.result())
    except Exception as exc:
        lg.error(exc)


    lg.info('Exiting client')

    sys.exit()

if __name__ == "__main__":
    main()
```



```
(snippets-env) J:\Education\Code\Python\Python-Snippets\programs\lan_client>python client.py
appLogger - 2020-09-08 12:40:54,413-11056-INFO-Client is connected to a chat server
a
appLogger - 2020-09-08 12:40:59,239-11056-INFO-Server: b
appLogger - 2020-09-08 12:41:02,308-11056-INFO-Closing client connection

appLogger - 2020-09-08 12:41:09,109-11056-ERROR-
[WinError 10054] An existing connection was forcibly closed by the remote host
appLogger - 2020-09-08 12:41:09,110-11056-INFO-None
appLogger - 2020-09-08 12:41:09,125-11056-INFO-Exiting client
```