Python snippets

Contents

[Digital Clock using python 3](#_Toc50462407)

[Python messenger using LAN 4](#_Toc50462408)

# 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:

"""

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