TypeError: a bytes-like object is required, not 'str'

Asked 5 years, 8 months ago Active 1 month ago Viewed 206k times

69

14

The following is the code that tries to modify the input supplied by a user by using sockets:

```
from socket import *

serverName = '127.0.0.1'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print (modifiedMessage)
clientSocket.close()
```

When I execute it and supply input the following error occurs:

```
Input lowercase sentence:fdsgfdf
Traceback (most recent call last):
  File "C:\srinath files\NETWORKS\UDPclient.py", line 6, in <module>
    clientSocket.sendto(message,(serverName, serverPort))
TypeError: a bytes-like object is required, not 'str'
```

What can I do to solve this?

python python-3.x sockets

Share Follow

edited Jun 12 '19 at 12:02
meager ♦
210k 38 307 315

asked Oct 7 '15 at 22:29
sri
691 1 5 3

4 The first argument (`message`) needs to be bytes, but you're passing a string. You should encode it before sending e.g. `message.encode('utf-8')` – mgilson Oct 7 '15 at 22:34

but the thing is i need to pass string to the server not byte – sri Oct 7 '15 at 22:57

from socket import * serverName = 'hostname' serverPort = 12000 clientSocket = socket(AF_INET, SOCK_DGRAM) message = input('Input lowercase sentence:') message.encode('utf-8') clientSocket.sendto(message,(serverName, serverPort)) modifiedMessage, serverAddress = clientSocket.recvfrom(2048) print (modifiedMessage) clientSocket.close() – sri Oct 7 '15 at 22:57

`clientSocket.sendto(message.encode('utf-8'), ...)` – mgilson Oct 7 '15 at 22:59

5 Answers

Active Oldest Votes

81

This code is good for Python 2. But in Python 3, results in bit encoding error. I was trying to make a simple TCP server and encountered the same problem. Encoding solves this. Try this with `sendto` command.

```
clientSocket.sendto(message.encode(),(serverName, serverPort))
```

Similarly you should use `.decode()` to receive the data on the UDP server side, if you want to print it exactly as it was sent.

Share Follow edited Apr 7 at 9:40 answered Dec 16 '15 at 13:44

Umair47
942 7 11

would encoding and decoding on both sides of the connection increase latency? If so would python2 actually be faster at sending data than python3 – Max Oct 12 '17 at 16:14

In my opinion, this will not affect the performance such as latency etc. However, I haven't tried experimented with it myself so I can't be sure about that. – Umair47 Oct 19 '17 at 17:32

30

Encoding and decoding can solve this in Python 3:

Client Side:

```
>>> host='127.0.0.1'
>>> port=1337
>>> import socket
>>> s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
>>> s.connect((host,port))
>>> st='connection done'
>>> byt=st.encode()
>>> s.send(byt)
15
>>>
```

Server Side:

```
>>> host=''
>>> port=1337
>>> import socket
>>> s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
>>> data.decode()

'connection done'
>>>
```

Share Follow

edited Jun 10 '19 at 16:35
Felipe Augusto
5,831 7 29 60

answered Mar 19 '18 at 11:58
Ahmed Motawea
301 3 3

2 See also nedbatchelder.com/text/unipain.html to understand why this changed in Python 3. – tripleee Aug 18 '19 at 7:03

13

A bit of encoding can solve this:

Client Side:

```
message = input("->")
clientSocket.sendto(message.encode('utf-8'), (address, port))
```

Server Side:

```
data = s.recv(1024)
modifiedMessage, serverAddress = clientSocket.recvfrom(message.decode('utf-8'))
```

Share Follow

edited Jun 5 '17 at 18:31

answered Jul 18 '16 at 16:25
Wlliam
165 4 11

1 Your client won't work: sendto requires 3 parameters, the message the server address and the related port – dlewin Jun 2 '17 at 14:41

Message, address, and port. Seems like 3 parameters to me and works on mine! :) – HotWheels Feb 26 '20 at 18:42

5

Simply replace message parameter passed in `clientSocket.sendto(message,(serverName, serverPort))` to `clientSocket.sendto(message.encode(),(serverName, serverPort))` . Then you would successfully run in in *python3*

Share Follow

edited Mar 10 '17 at 21:12
Evgeniy Mironov
709 5 22

answered Mar 10 '17 at 18:11
TestStart
93 1 9

3

(where `my_string` is the string you're passing to a function/method).

The `encode` method of `str` objects returns the encoded version of the string as a `bytes` object which you can then use. In this specific instance, socket methods such as `.send` *expect a bytes object* as the data to be sent, *not a string object*.

Since you have an object of type `str` and you're passing it to a function/method that expects an object of type `bytes`, an error is raised that clearly explains that:

```
TypeError: a bytes-like object is required, not 'str'
```

So the `encode` method of strings is needed, applied on a `str` value and returning a `bytes` value:

```
>>> s = "Hello world"
>>> print(type(s))
<class 'str'>
>>> byte_s = s.encode()
>>> print(type(byte_s))
<class 'bytes'>
>>> print(byte_s)
b"Hello world"
```

Here the prefix `b` in `b'Hello world'` denotes that this is indeed a bytes object. You can then pass it to whatever function is expecting it in order for it to run smoothly.

Share Follow

answered Nov 25 '17 at 13:07
Dimitris Fasarakis Hilliard
120k 27 228 224