Network Data and Network Errors

Created By Shreen Khalaf

Bytes and Strings

▶ In Python, you will normally represent bytes in one of two ways:

   ▶ **Integer** whose value happens to be between 0 and 255

   ▶ **a length-1 byte string** where the byte is the single value that it contains.

   ▶ You can type a byte-valued number using any of the typical bases supported in Python source code—binary, octal, decimal, and hexadecimal

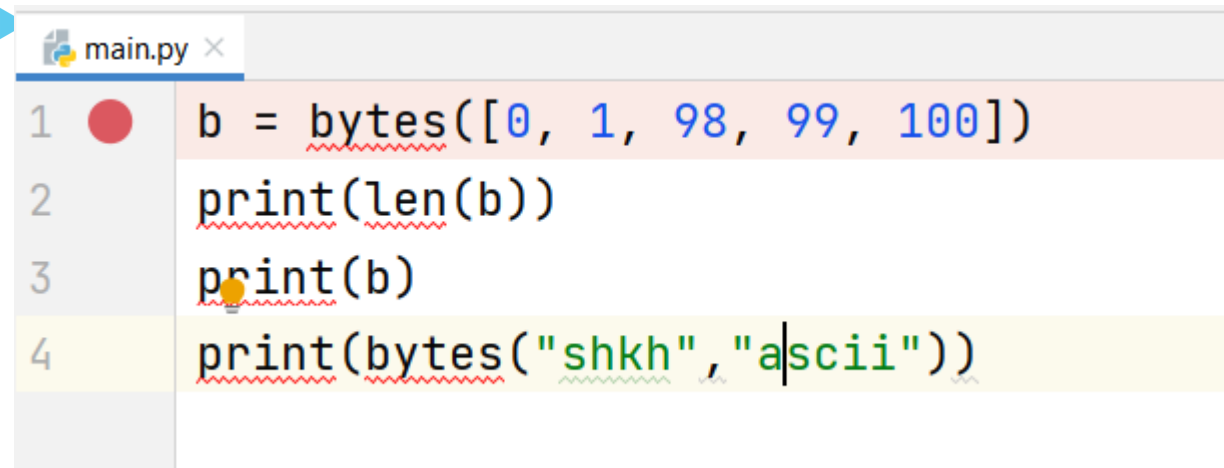   ▶ 0b1100010 == 0o142 == 98 == 0x62

Python bytes() Function

- The **bytes()** function returns a bytes object.
- It can convert objects into bytes objects
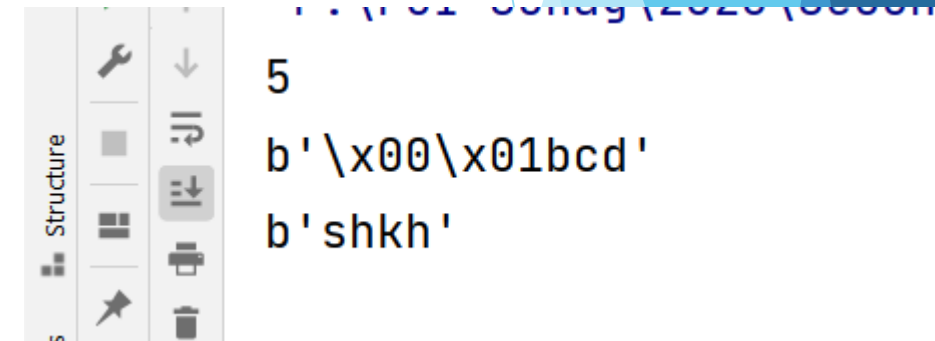- **Syntax**
  - bytes(*x, encoding, error*)

Parameter Values

Parameter	Description
x	A source to use when creating the bytes object.
	If it is an integer, an empty bytes object of the specified size will be created.
	If it is a String, make sure you specify the encoding of the source.
encoding	The encoding of the string
error	Specifies what to do if the encoding fails.

Bytes and Strings

▶ convert a list of such numbers to a byte string by passing them to the bytes()

▶ b = bytes([0, 1, 98, 99, 100])

```python
b = bytes([0, 1, 98, 99, 100])
print(len(b))
print(b)
print(bytes("shkh","ascii"))
```

```
5
b'\x00\x01bcd'
b'shkh'
```

Character Strings

- Encoding characters means turning a string of real Unicode characters into bytes that can be sent out into the real world outside your Python program.

- The encode() method encodes the string, using the specified encoding. If no encoding is specified, UTF-8 will be used

- https://www.w3schools.com/charsets/ref_html_utf8.asp

encode()　method

- **Syntax**
  - encode(encoding=*encoding*, errors=*errors*)

Parameter Values

Parameter	Description
encoding	Optional. A String specifying the encoding to use. Default is UTF-8
errors	Optional. A String specifying the error method. Legal values are:

`'backslashreplace'`	- uses a backslash instead of the character that could not be encoded
`'ignore'`	- ignores the characters that cannot be encoded
`'namereplace'`	- replaces the character with a text explaining the character
`'strict'`	- Default, raises an error on failure
`'replace'`	- replaces the character with a questionmark
`'xmlcharrefreplace'`	- replaces the character with an xml character

Character Strings

- Decoding byte data means converting a byte string into real characters.

- **Syntax**

  - Str.decode(encoding='UTF-8',errors='strict')

- UTF (Unicode Transformation Format).

- The Unicode Standard is implemented in HTML, XML, JavaScript, E-mail, PHP, Databases and in all modern operating systems and browsers.

Charset	Description
UTF-8	A variable-length character encoding (1 to 4 bytes long). UTF-8 is backwards compatible with ASCII and the preferred encoding for e-mail and web pages.
UTF-16	A variable-length character encoding. UTF-16 is used in all major operating systems like Windows, IOS, and Unix.

struct module
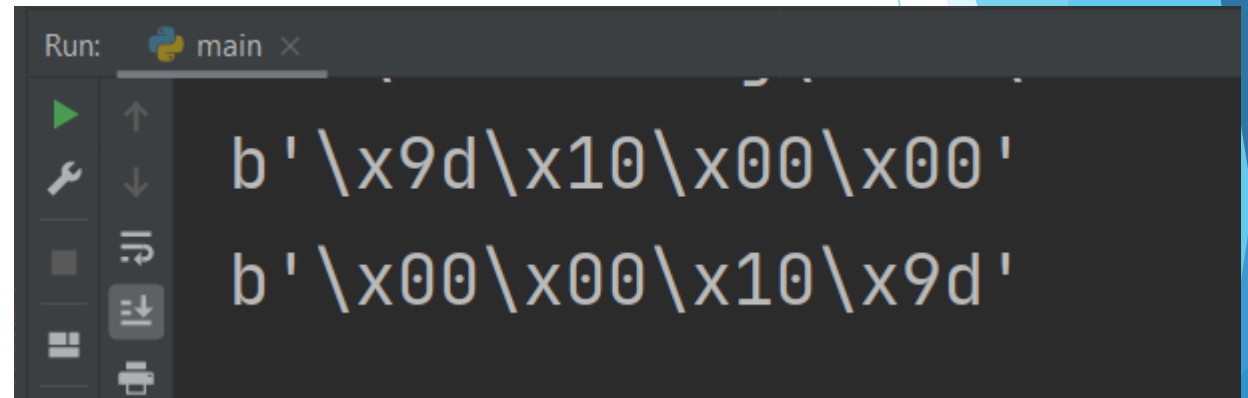
- **struct module**, provides a variety of operations for converting data to and from popular binary formats.

- **struct.pack(*format, v1, v2, ...*)**

  - Return a bytes object containing the values *v1*, *v2*, ... packed according to the format string *format*.

- **struct.unpack(*format*, *buffer*)**

  - which converts the binary data back to Python numbers.

Created By Shreen Khalaf

struct module

```python
import struct

print(struct.pack('<i', 4253))
print(struct.pack('>i', 4253))
```



```
Run:      main  ×

b'\x9d\x10\x00\x00'

b'\x00\x00\x10\x9d'
```

formatting code 'i', which uses four bytes to store an integer, and this leaves the two upper bytes zero for a small number like 4253.

Framing and Quoting

▶ **How to delimit your messages so that the receiver can tell where one message ends and the next one begins ??**

1. The receiver call recv() repeatedly until the call finally returns an empty string

2. Use fixed-length messages

3. Delimit your messages with special characters.

4. To prefix each message with its length

5. Instead of sending just one, try sending several blocks of data that are each prefixed with their length.

The receiver call recv() repeatedly until the call finally returns an empty string

```python
message = b''
while True:
    more = sc.recv(8192) # arbitrary value of 8k
    if not more: # socket has closed when recv() returns ''
        print('Received zero bytes - end of file')
        break
    print('Received {} bytes'.format(len(more)))
    message += more
print('Message:\n')
print(message.decode('ascii'))
```

Server

```python
import socket
sok=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
host=socket.gethostname()
port=12345
serv_add=(host,port)
sok.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
sok.bind(serv_add)
print('waiting for client')
sok.listen()
conn,add=sok.accept()
conn.shutdown(socket.SHUT_WR)
msg=b''
while True:
    data=conn.recv(16)
    if not data:
        print('no more data')
        break
    print(data)
    msg=msg+data
print(msg.decode())
```

Client

```python
import socket
sok=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
host=socket.gethostname()
port=12345
serv_add=(host,port)
sok.connect(serv_add)
print('Connect to server')
sok.shutdown(socket.SHUT_RD)
sok.sendall('Client: Hello'.encode())
sok.sendall('Client: test send message'.encode())
sok.sendall("Client: End".encode())
print('data Sent')
```

Use fixed-length messages

```python
def recvall(sock, length):
    data = ''
    while len(data) < length:
        more = sock.recv(length - len(data))
        if not more:
            raise
        data += more
    return data
```

How to delimit your messages so that the receiver can tell where one message ends and the next one begins ??

3. **Delimit your messages with special characters.**

   ▶ The receiver would wait in a recv() loop until the reply string it was accumulating finally contained the delimiter indicating the end-of-message.

4. **To prefix each message with its length**

   ▶ The length read and decoded, then

   ▶ The receiver can enter a loop and call recv() repeatedly until the whole message has arrived (use recvall(sock, length))

Instead of sending just one, try sending several blocks of data that are each prefixed with their length.

▶ **For read define get_block(sock) method**

   ▶ **Read length struct size and unpack to convert it to number**

     header_struct = struct.Struct('!I') # messages up to 2**32 - 1 in length

     len= recvall(sock, header_struct.size)

     (block_length,) = header_struct.unpack(len)

   ▶ **Read data using recvall(sok,len)**

     recvall(sock, block_length)

Instead of sending just one, try sending several blocks of data that are each prefixed with their length.

- **To write data define put block**

```python
import  struct
header_struct = struct.Struct('!I')
def put_block(sock, message):
    block_length = len(message)
    sock.send(header_struct.pack(block_length))
    sock.send(message)
```

Server

```python
import socket, struct
header_struct = struct.Struct('!I') # messages up to 2**32 - 1 in length
def server(address):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(address)
    sock.listen(1)
    print('Listening at', sock.getsockname())
    sc, sockname = sock.accept()
    print('Accepted connection from', sockname)
    sc.shutdown(socket.SHUT_WR)
    while True:
        block = get_block(sc)
        if not block:
            break
        print('Block says:', repr(block))
    sc.close()
    sock.close()
```

Get_block method

```python
def get_block(sock):
    data = sock.recv( header_struct.size)
    (block_length,) = header_struct.unpack(data)
    return recvall(sock, block_length)


def recvall(sock, length):
    data = ''
    while len(data) < length:
        more = sock.recv(length - len(data))
        print(data)
        if not more:
            raise
        data += more.decode()
    return data
server(('127.0.0.1',12346))
```

Client


```python
import socket, struct
header_struct = struct.Struct('!I')
def client(address):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STR
    sock.connect(address)
    sock.shutdown(socket.SHUT_RD)
    put_block(sock, b'Beautiful is better than ugly.')
    put_block(sock, b'Explicit is better than implicit.'
    put_block(sock, b'Simple is better than complex.')
    put_block(sock, b'')
    sock.close()

def put_block(sock, message):
    block_length = len(message)
    sock.send(header_struct.pack(block_length))
    sock.send(message)

client(('127.0.0.1', 12346))
```

repr() method

- **repr()**
- The repr() function returns a printable representation of the given object.

Thank You

Created By Shreen Khalaf