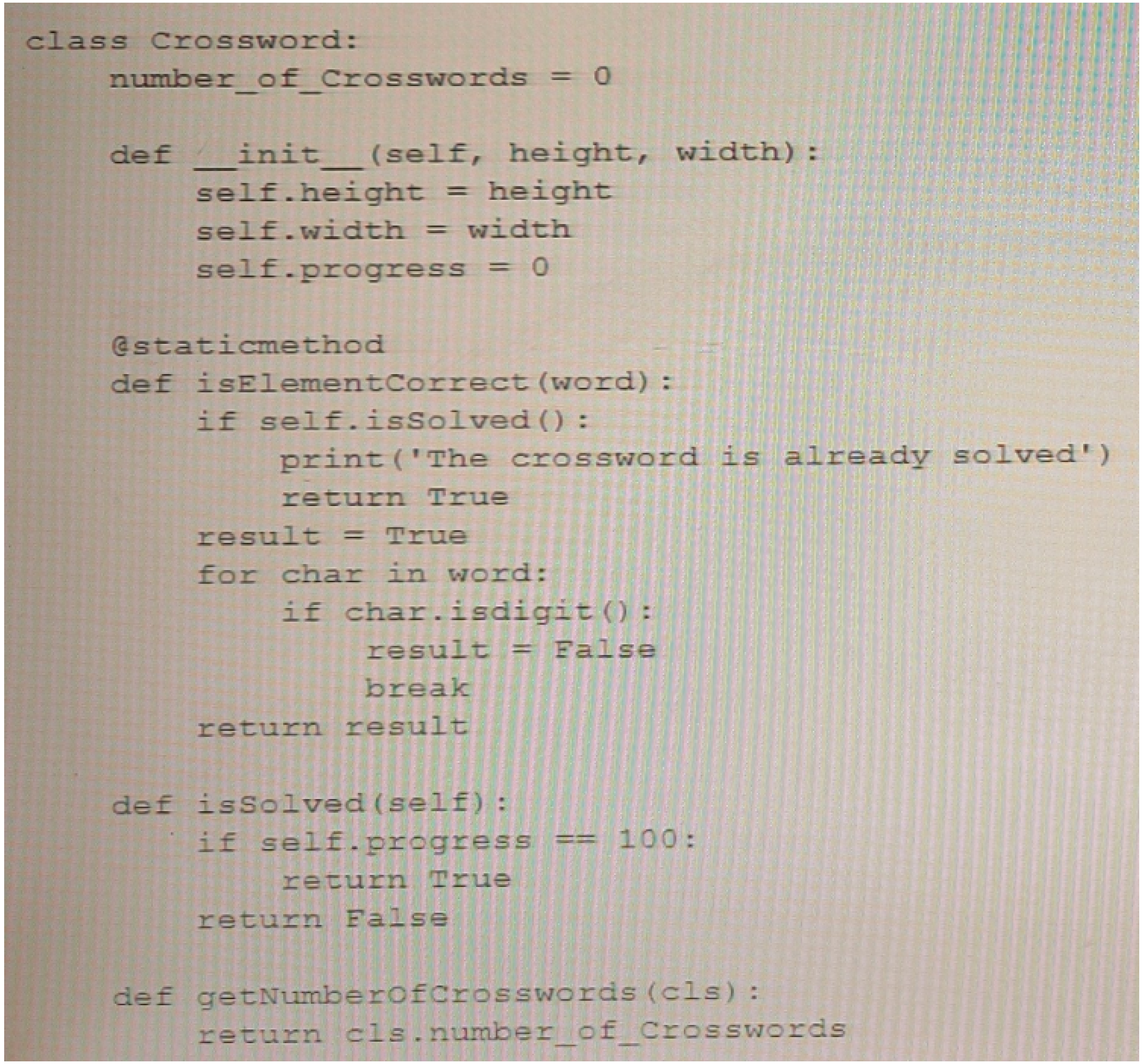
Question 1

Question Type: MultipleChoice

Analyze the following snippet and decide whether the code is correct and/or which method should be distinguished as a class method.



```
class Crossword:
    number_of_Crosswords = 0

    def __init__(self, height, width):
        self.height = height
        self.width = width
        self.progress = 0

    @staticmethod
    def isElementCorrect(word):
        if self.isSolved():
            print('The crossword is already solved')
            return True
        result = True
        for char in word:
            if char.isdigit():
                result = False
                break
        return result

    def isSolved(self):
        if self.progress == 100:
            return True
        return False

    def getNumberOfCrosswords(cls):
        return cls.number_of_Crosswords
```

Options:

A- There is only one initializer, so there is no need for a class method.

B- The getNumberofCrosswords () method should be decorated With @classmethod.

C- The code is erroneous.

D- The gexNumberOfcrosswords () and issrived methods should be decorated with @classzoechod.

Answer:

B

Explanation:

The correct answer isB. The getNumberofCrosswords() method should be decorated with @classmethod. In the given code snippet, thegetNumberofCrosswordsmethod is intended to be a class method that returns the value of thenumberofcrosswordsclass variable. However, the method is not decorated with the@classmethoddecorator and does not take aclsparameter representing the class itself. To makegetNumberofCrosswordsa proper class method, it should be decorated with@classmethodand take aclsparameter as its first argument.

1. ThegetNumberofCrosswords()method should be decorated with@classmethod.

This is because thegetNumberofCrosswords()method is intended to access the class-level variablenumberofcrosswords, but it is defined as an instance method, which requires an instance of the class to be created before it can be called. To make it work as a class-level method, you can define it as a class method by adding the@classmethoddecorator to the function.

Here's an example of how to definegetNumberofCrosswords()as a class method:

```
class Crossword:

numberofcrosswords = 0

def __init__(self, author, title):

self.author = author

self.title = title

Crossword.numberofcrosswords += 1

@classmethod

def getNumberofCrosswords(cls):

return cls.numberofcrosswords
```

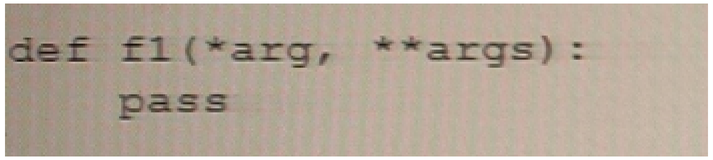
In this example,getNumberofCrosswords()is defined as a class method using the@classmethoddecorator, and theclsparameter is used to access the class-level variablenumberofcrosswords.

Official Python documentation on Classes:https://docs.python.org/3/tutorial/classes.html

Question 2

Question Type: MultipleChoice

Analyze the following snippet and select the statement that best describes it.



```
def f1(*arg, **args):
    pass
```

Options:

A- The code is syntactically correct despite the fact that the names of the function parameters do not follow the naming convention

B- The *arg parameter holds a list of unnamed parameters

C- The code is missing a placeholder for unnamed parameters.

D- The code is syntactically incorrect - the function should be defined as def f1 (*args, **kwargs) :

Answer:

B

Explanation:

The provided code snippet defines a functionf1that accepts variable-length arguments using the*argsand**kwargssyntax. The*argsparameter allows for an arbitrary number of unnamed arguments to be passed to the function as a tuple, while the**kwargsparameter allows for an arbitrary number of named arguments to be passed to the function as a dictionary.

Therefore, the correct statement that best describes the code is:

1. The*argsparameter holds a list of unnamed parameters, while the**kwargsparameter holds a dictionary of named parameters.

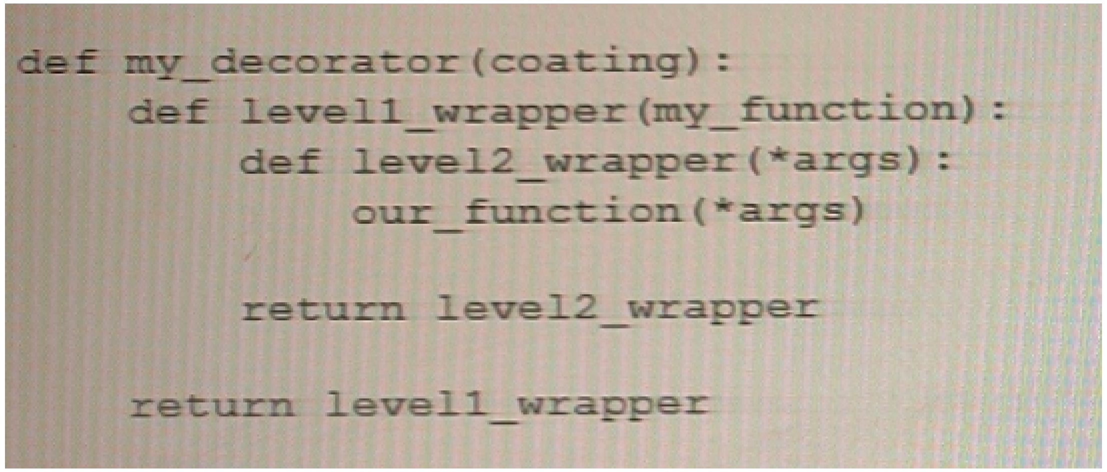
Official Python documentation on Function definitions:https://docs.python.org/3/tutorial/controlflow.html#defining-functions

Thearg parameter holds a list of unnamed parameters. In the given code snippet, thef1function takes two arguments:*argand**kwarg. The*argsyntax in the function signature is used to pass a variable number of non-keyword (positional) arguments to the function. Inside the function,argis a tuple containing the positional arguments passed to the function. The**kwargsyntax in the function signature is used to pass a variable number of keyword arguments to the function. Inside the function,kwargis a dictionary containing the keyword arguments passed to the function.

Question 3

Question Type: **MultipleChoice**

Analyze the following function and choose the statement that best describes it.



```
def my_decorator(coating):
    def level1_wrapper(my_function):
        def level2_wrapper(*args):
            our_function(*args)

        return level2_wrapper

    return level1_wrapper
```

Options:

A- It is an example of a decorator that accepts its own arguments.

B- It is an example of decorator stacking.

C- It is an example of a decorator that can trigger an infinite recursion.

D- The function is erroneous.

Answer:

A

Explanation:

In the given code snippet, therepeatfunction is a decorator that takes an argumentnum_timesspecifying the number of times the decorated function should be called. Therepeatfunction returns an inner functionwrapper_repeatthat takes a functionfuncas an argument and returns another inner functionwrapperthat callsfuncnum_timestimes.

The provided code snippet represents an example of a decorator that accepts its own arguments. The@decorator_functionsyntax is used to apply thedecorator_functionto thesome_functionfunction. Thedecorator_functiontakes an argumentarg1and defines an inner functionwrapper_functionthat takes the original functionfuncas its argument. Thewrapper_functionthen returns the result of callingfunc, along with thearg1argument passed to thedecorator_function.

Here is an example of how to use this decorator withsome_function:

@decorator_function('argument 1')

def some_function():

return 'Hello world'

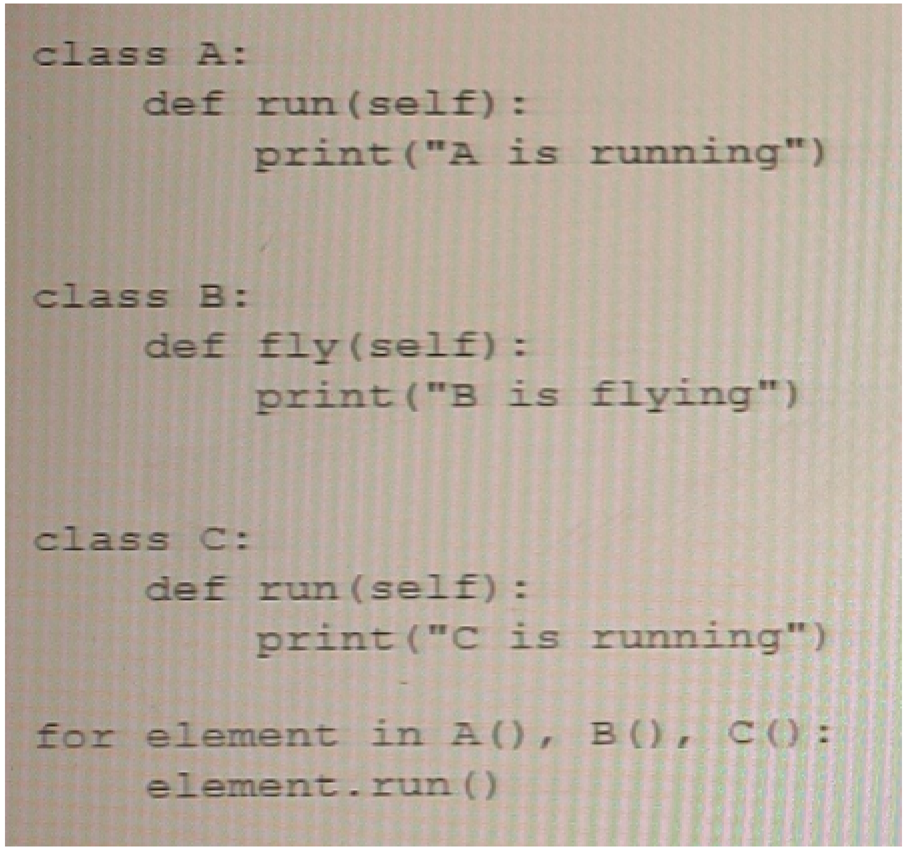
Whensome_functionis called, it will first be passed as an argument to thedecorator_function. Thedecorator_functionthen adds the string'argument 1'to the result of callingsome_function()and returns the resulting string. In this case, the final output would be'Hello world argument 1'.

Official Python documentation on Decorators:https://docs.python.org/3/glossary.html#term-decorator

Question 4

Question Type: MultipleChoice

The following snippet represents one of the OOP pillars Which one is that?



```
class A:
    def run(self):
        print("A is running")


class B:
    def fly(self):
        print("B is flying")


class C:
    def run(self):
        print("C is running")

for element in A(), B(), C():
    element.run()
```

Options:

A- Serialization

B- Inheritance

C- Encapsulation

D- Polymorphism

Answer:

C

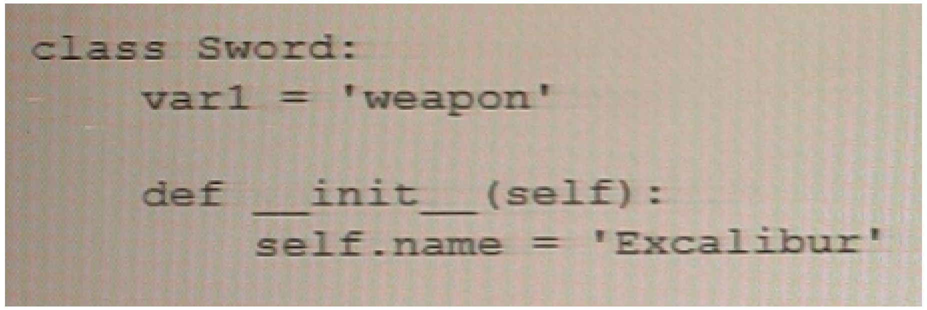
Explanation:

The given code snippet demonstrates the concept of encapsulation in object-oriented programming. Encapsulation refers to the practice of keeping the internal state and behavior of an object hidden from the outside world and providing a public interface for interacting with the object. In the given code snippet, the__init__andget_balancemethods provide a public interface for interacting with instances of theBankAccountclass, while the__balanceattribute is kept hidden from the outside world by using a double underscore prefix.

Question 5

Question Type: MultipleChoice

Analyze the following snippet and choose the best statement that describes it.



```
class Sword:
    var1 = 'weapon'

    def __init__(self):
        self.name = 'Excalibur'
```

Options:

A- self. name is the name of a class variable.

B- varl is the name of a global variable

C- Excalibur is the value passed to an instance variable

D- Weapon is the value passed to an instance variable

Answer:

C

Explanation:

The correct answer isC. Excalibur is the value passed to an instance variable. In the given code snippet,self.nameis an instance variable of theSwordclass. When an instance of theSwordclass is created withvarl = Sword('Excalibur'), the value'Excalibur'is passed as an argument to the__init__method and assigned to thenameinstance variable of thevarlobject.

The code defines a class calledSwordwith an__init__method that takes one parametername. When a new instance of theSwordclass is created withvarl = Sword('Excalibur'), the value of the'Excalibur'string is passed as an argument to the__init__method, and assigned to theself.nameinstance variable of thevarlobject.

Official Python documentation on Classes:https://docs.python.org/3/tutorial/classes.html

Question 6

Question Type: MultipleChoice

Select the true statement about composition

Options:

A- Composition extends a class's capabilities by adding new components and modifying the existing ones.

B- Composition allows a class to be projected as a container of different classes

C- Composition is a concept that promotes code reusability while inheritance promotes encapsulation.

D- Composition is based on the has a relation: so it cannot be used together with inheritance.

Answer:

B

Explanation:

Composition is an object-oriented design concept that models ahas-a relationship. In composition, a class known ascompositecontains an object of another class known ascomponent.In other words, a composite class has a component of another class1.

1. Composition allows a class to be projected as a container of different classes.

Composition is a concept in Python that allows for building complex objects out of simpler objects, by aggregating one or more objects of another class as attributes. The objects that are aggregated are generally considered to be parts of the whole object, and the containing object is often viewed as a container for the smaller objects.

In composition, objects are combined in a way that allows for greater flexibility and modifiability than what inheritance can offer. With composition, it is possible to create new objects by combining existing objects, by using a container object to host other objects. By contrast, with inheritance, new objects extend the behavior of their parent classes, and are limited by that inheritance hierarchy.

Official Python documentation on Composition:https://docs.python.org/3/tutorial/classes.html#composition

GeeksforGeeks article on Composition vs Inheritance:https://www.geeksforgeeks.org/composition-vs-inheritance-python/

Real Python article on Composition and Inheritance:https://realpython.com/inheritance-composition-python/

Question 7

Question Type: MultipleChoice

What is true about the unbind () method? (Select two answers.)

Options:

A- It is invoked from within the events object

B- It is invoked from within a widget's object

C- It needs a widget's object as an argument

D- It needs the event name as an argument

Answer:

B, D

Explanation:

Option B is true because theunbind()method is invoked from within a widget's object1.

Option D is true because theunbind()method needs the event name as an argument1.

Theunbind()method in Tkinter is used to remove a binding between an event and a function. It can be invoked from within a widget's object when a binding is no longer needed. The method requires the event name as an argument to remove the binding for that specific event. For example:

button = tk.Button(root, text='Click me')

button.bind('<Button-1>', callback_function) # bind left mouse click event to callback_function

button.unbind('<Button-1>') # remove the binding for the left mouse click event

Question 8

Question Type: **MultipleChoice**

Select the true statements about the connection-oriented and connectionless types of communication. (Select two answers.)

Options:

A- In the context of TCP/IP networks, the communication side that initiates a connection is called the client, whereas the side that answers the client is called the server

B- Connectionless communications are usually built on top of TCP

C- Using walkie-talkies is an example of a connection-oriented communication

D- A phone call is an example of a connection-oriented communication

Answer:

A, D

Explanation:

1. In the context of TCP/IP networks, the communication side that initiates a connection is called the client, whereas the side that answers the client is called the server.

This statement is true because TCP/IP networks use a client-server model to establish connection-oriented communications. The client is the device or application that requests a service or resource from another device or application, which is called the server. The server responds to the client's request and provides the service or resource. For example, when you browse a website using a web browser, the browser acts as a client and sends a request to the web server that hosts the website.The web server acts as a server and sends back the requested web page to the browser1.

2. Connectionless communications are usually built on top of TCP.

This statement is false because TCP (Transmission Control Protocol) is a connection-oriented protocol that requires establishing and terminating a connection before and after sending data. Connectionless communications are usually built on top of UDP (User Datagram Protocol), which is a connectionless protocol that does not require any connection setup or teardown.UDP simply sends data packets to the destination without checking if they are received or not2.

3. Using walkie-talkies is an example of a connection-oriented communication.

This statement is false because using walkie-talkies is an example of a connectionless communication. Walkie-talkies do not establish a dedicated channel or connection between the sender and receiver before transmitting data. They simply broadcast data over a shared frequency without ensuring that the receiver is ready or available to receive it.The sender does not know if the receiver has received the data or not3.

4. A phone call is an example of a connection-oriented communication.

This statement is true because a phone call is an example of a connection-oriented communication. A phone call requires setting up a circuit or connection between the caller and callee before exchanging voice data. The caller and callee can hear each other's voice and know if they are connected or not.The phone call also requires terminating the connection when the conversation is over4.

1: https://www.techtarget.com/searchnetworking/definition/client-server2: https://www.javatpoint.com/connection-oriented-vs-connectionless-service3: https://en.wikipedia.org/wiki/Walkie-talkie4: https://en.wikipedia.org/wiki/Telephone_call

A is true because in the context of TCP/IP networks, the communication side that initiates a connection is called the client, and the side that answers the client is called the server. This is the basis for establishing a connection-oriented communication.

D is true because a phone call is an example of a connection-oriented communication. Like TCP/IP, a phone call establishes a connection between two devices (in this case, two phones) before communication can occur.

A is true because in the context of TCP/IP networks, the communication side that initiates a connection is called the client, and the side that answers the client is called the server. This is the basis for establishing a connection-oriented communication.

D is true because a phone call is an example of a connection-oriented communication. Like TCP/IP, a phone call establishes a connection between two devices (in this case, two phones) before communication can occur.

B is false because connectionless communications are usually built on top of UDP, not TCP. UDP is a connectionless protocol that does not establish a connection before sending data.

C is false because using walkie-talkies is an example of a connectionless communication. Walkie-talkies do not establish a connection before communication begins, and messages are simply broadcasted to all devices within range.

Here is a sample code in Python using thesocketmodule to create a TCP server and client to demonstrate the connection-oriented communication:

Server-side code:

```
import socket

HOST = '127.0.0.1'

PORT = 8080

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:

s.bind((HOST, PORT))

s.listen()
```

```
conn, addr = s.accept()

with conn:

print('Connected by', addr)

while True:

data = conn.recv(1024)

if not data:

break

conn.sendall(data)
```

Client-side code:

```
import socket

HOST = '127.0.0.1'

PORT = 8080

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:

s.connect((HOST, PORT))

s.sendall(b'Hello, world')
```

```
data = s.recv(1024)

print('Received', repr(data))
```

The server listens for incoming connections on port 8080, and when a connection is established, it prints the address of the client that has connected. The server then continuously receives data from the client and sends it back to the client until the connection is closed.

The client establishes a connection with the server and sends the message 'Hello, world' encoded as bytes. It then waits for a response from the server and prints the data it receives.