I.J. Information Technology and Computer Science, **2023, 4, 50-62**
Published Online on August 8, 2023 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2023.04.05

Implementation of Python Interoperability in Java through TCP Socket Communication

Bala Dhandayuthapani V.
Department of Information Technology, University of Technology and Applied Sciences - Shinas, Oman
E-mail: bala.veerasamy@shct.edu.om, dhanssoft@gmail.com
ORCID iD: https://orcid.org/0000-0002-8310-0642

Received: 21 December 2022; Revised: 11 March 2023; Accepted: 19 May 2023; Published: 08 August 2023

Abstract: Programming language interoperability is highly desirable for a variety of reasons, such as the fact that if a programmer implements specific functionality that has previously been implemented in another language, the software component can simply be reused. Because they are particularly well-suited and efficient at implementing features, certain languages regularly arise to handle issue areas. There are numerous third-party programs available for a variety of languages. When programmers have experience with and preferences for several programming languages, collaboration on complex projects is easier. A range of techniques and methods have been used to handle various cross-language communication challenges. The importance of interoperability and cross-language communication between Java and Python via socket programming is examined in this research article through an empirical model of different execution environment paradigms that can help guide the development of improved approaches for integrating Python libraries with Java without the need for extra libraries or third-party libraries. The interoperability strategy benefits from the quality and availability of Python libraries in Java by cutting down on development time, maintenance needs, general usability, upkeep, and system integration without incurring additional costs. It is versatile to use this interoperability strategy since identical scripts are run in Java client contexts in the same way that they were used in Python. There are different Python modules used in the research article to exemplify and evaluate the expressions, built-in functions, strings, collections, data exploration, statistical data analysis using NumPy, SciPy, and Pandas, and Scikit-Learn for machine learning with linear regression.

Index Terms: Java, Interoperability, Libraries, Python, Socket Programming.

1. Introduction

Programming language interoperability refers to the capacity of codes written in two or more programming languages to communicate with one another as components of the same system [1, 2]. Interoperability refers to the ability of at least two systems or applications to share and use data [3]. On the other hand, cloud interoperability is the capacity to share data between two or more cloud services in line with a strategy to achieve results. Language interoperability is highly desirable for a variety of reasons. For example, if a programmer must implement specific functionality that has previously been built in another language, the matching software component can easily be reused. Because they are particularly well-suited and efficient at implementing features, some languages have evolved specifically to address specific problem domains. There are numerous third-party programs available for a variety of languages. Furthermore, every coder has a preferred language in which they excel in terms of competence and efficiency. The use of hundreds of programming languages and the continual development of new ones necessitate language interoperability. When working on complex projects, programmers from various backgrounds and programming languages can interact more easily.

Interoperability is achieved by translating the interface standards into the desired programming language. Language mapping [4] defines the representation in the target computer language of an a priori agreed-upon collection of data types. The developer must adhere to this convention. However, achieving programming language compatibility will necessitate additional procedures. Interoperability is necessary for companies to use Python and Java for machine learning, data analysis, legacy systems, and multi-language environments. It is determined by the use case, such as high performance or scalability, and the specific interoperability requirements are determined by the use case.

Python is a popular data analytics programming language due to its ease of use, vast library ecosystem, and active community. Java is more verbose and has a steeper learning curve. Python has several compiled and optimized libraries that are faster than Java. Java is better suited for large-scale data analytics applications that require high performance and scalability. The benefits of interoperability allow programmers to benefit from both languages, such as Python scripting's

ease and Java's strength. This enables system integration, which is particularly useful for integrating libraries. Depending on the application requirements, there are numerous choices for interoperability, such as employing Jython, Py4J, or other ways that can result in applications that are more effective and powerful by leveraging the strengths of both languages.

Because data conversion and communication between the two languages are required, the downside may result in performance costs, particularly in applications that require high performance or low latency. Disparities in data formats and object models, for example, can be a substantial impediment to interoperability and may necessitate the use of specific tools or workarounds to overcome them. Integrating two distinct programming languages can be challenging and costly since it necessitates specialized knowledge and equipment. The availability of libraries, particularly for certain use cases, may provide a hurdle to interoperability. The following subsections covered the problem statement, assumption, significance, scope, limitations, and hypotheses.

1.1. Statement of Problem

Businesses that utilize Python for machine learning and data analysis and Java for production systems confront a hurdle when attempting to merge the two languages. Because Java and Python are incompatible, it is difficult to interchange code, build end-to-end pipelines, and deploy machine learning models in real-world applications. These businesses desire a solution that enables smooth Java and Python interoperation to take advantage of the advantages of both languages and construct more efficient and scalable systems. Jython, CPython, Py4J, and other wrapper libraries, which are required for compatibility due to incompatibilities between the Python and Java ecosystems, may not support all Python modules and libraries. This implementation paradigm provides Python compatibility in Java without the requirement for additional libraries via TCP socket communication.

1.2. Assumptions

The choice of interoperability techniques may vary depending on the application's specific requirements, such as performance, convenience of use, and the availability of Python libraries. Because of the necessity for data conversion and communication between the two languages, interoperability might impose performance overhead. This might be an issue in applications requiring high performance or low latency. Interoperability can be hampered by compatibility difficulties such as discrepancies in data types and object models between the two languages. Python and Java can be readily merged utilizing TCP sockets and simple code to benefit from the advantages of the Python language over Java.

1.3. Significance

Python is well-known for its ease of use, flexibility, and huge library ecosystem, whereas Java is well-known for its scalability, dependability, and security. Because Python is compatible with Java, developers can utilize it to create more effective and efficient applications. Python is widely used for machine learning and data analysis, whereas Java is widely utilized for constructing scalable production systems. Organizations may design end-to-end pipelines that incorporate machine learning models produced in Python and deploy them in production systems created in Java by enabling interoperability between the two languages.

1.3. Scope and limitations

Scope: Creating end-to-end pipelines that incorporate Python-based machine learning models and deploying them in Java-based production systems. Integrating legacy Java-based systems with newer Python-based ones. Promoting collaboration across teams that employ various programming languages. Using both languages' strengths to create more effective and efficient applications. Using Python to enable data analysis and Java to create an enterprise dashboard.

Limitations: Interoperability between Python and Java may result in performance overhead due to communication delays because data conversion and communication between the two languages are necessary. This typically involves delivering messages in the form of instruction codes and data between languages that may be extremely distinct, which causes major problems. Python has a deep ecosystem of modules for machine learning and data analysis, but communicating the data visualization may be difficult.

1.4. Hypotheses

When compared to alternative methods, Jython or Py4J for Python interoperability with Java can significantly improve data conversion and communication between the two languages. The availability of Python and Java libraries can have a significant impact on how straightforward and successful it is to combine the two, with fewer options accessible for some specialized use cases. Python and Java can be used together to create a powerful combination of scripting and object-oriented programming, resulting in more flexible and powerful applications. These hypotheses can be evaluated through empirical investigations and data analysis on Python interoperability with Java, and they can assist in the development of improved techniques for integrating Python libraries with Java languages. The following sections address the related literature review, methodology, results, and discussion. Finally, brief conclusions are provided.

In Section 2, It is found and learned the literature review of the related works. The most common method for interoperability used the wrapper libraries in different programming languages such as Jython, Py4J, and JPype and in web services-based interoperability on Simple Object Access Protocol (SOAP) or Representational State Transfer (REST). In Section 3, the methodology is introduced with different execution environment paradigms which are a local

server with a local client, a remote server with a local client, a cloud server with an endpoint client, and a cloud server with a cloud client. The execution model preferred Python as a server and Java as a client. In Section 4, the result and discussion examined the interoperability between the Python server and Java client. It is executed and showed the result for expression evaluations, Python built-in functions, Python string methods, Python collections, and Python data analysis modules such as NumPy, SciPy, and Pandas. In Section 5, the conclusion is given briefly. This research is an empirical model that proves Python code execution in Java without having additional libraries. However, Python should add all the libraries before using it in Java. It also described the boundary of completed research and future directions.

2. Related Works

There are different programming languages often used by developers to implement various functionalities in a single project. The second study question investigates language interoperability to define which languages are often utilized in multilingual projects. If two languages are utilized in the same project, we presume that they are interoperable [5]. Since the creation of the second programming language [6, 7], interoperability between languages has been a concern. Virtual machines (VMs) created to combine languages, including the JVM (Java Virtual Machine) and CLR, as well as language-independent object models like COM (Component Object Model) and CORBA (Common Object Request Broker Architecture), have all been used as solutions. (Common Language Runtime). A single language being the ideal tool for a whole program is less likely than before, as software grows refined, and hardware becomes less uniform. A fresh wave of intriguing solutions could emerge as current compilers become more flexible.

Language interoperability in a Multilanguage [8] context is a difficult issue for both legacy software users and producers of new large numerical software packages. Given the variety of programming languages used in scientific computing, library authors may discover that selecting one implementation language over another significantly limits the reuse of their numerical software. The key challenge was identified as researching strategies to satisfy the full interoperability need [9] for information systems, with a focus on solutions specifically for the IoT, software ecosystem, and cloud computing contexts. There are also application domains for banking systems, e-government, and health care.

Language interoperability is achieved with minimal work necessary for each language's implementation and with few limits on how each language is gathered on the platform. It describes a highly customizable yet successful system for hosting many programming languages on an object-oriented virtual computer [10]. The idea is to give each language a distinct perspective on a single class by putting language-specific wrapper methods in the class interface. This technique can take advantage of both Java's static virtual machine and Smalltalk's dynamic virtual machine. The language interoperability approach was developed on a virtual machine prototype for embedded systems based on the Smalltalk object model, and it supports embedded versions of the Smalltalk, Java, and BETA programming languages.

Virtual machines and markup languages each offer advantages and downsides in specific application contexts [11]. Virtual machines are far more viable for systems being built from the ground up when all linguistic decisions are made by the developers. The markup languages are better suited to dealing with existing or legacy systems where there is too much code to make rewriting a viable option or where parts of the source code are just unavailable, potentially due to collaboration with third-party software. Recompiling existing software is also not possible if the existing system cannot target a certain virtual machine, possibly due to a lack of a compiler from that language to that VM.

The method was developed for easing the pain of language interoperability in those systems by automating the process of generating binding code between various languages [12]. The suggested method systematically generates the Application Programming Interface (API) in each relevant language using the information gathered from the Interface Description Language (IDL). As a result, code created in one language can interact with code created in others without any issues. The experiment's findings proved that the produced code generator has increased the multi-language software systems' scalability, modularity, and stability.

The interoperability resulted from numerous activities occurring at multiple levels [13]. Application-level issues have to do with the ability to connect to the network. Concerns about service levels are related to services that can be compared to resolve compatibility problems. Concerns about the process level are those created for a system. Redesigned as conceptual, dynamic, pragmatic, semantic, syntactic, and technical, the level of conceptual interoperability model (LCIM) for blockchain is a level of interoperability model. An application program [14] must be able to easily share data with the programs around it to be used to its fullest potential. To carry out this, a method has been created to make application software interoperable based on the analysis of their data, using the specific characteristics of the data to classify them and make them suitable for interchange.

In the context of cloud computing [3], the capability to transfer a system from one Cloud platform to another is referred to as portability [15]. Each Cloud provider has their own set of current standards, data formats, and APIs. (APIs). These components must work together for Cloud services to be interoperable. Cloud APIs define how software programs communicate with a platform that is based in the cloud and on which they can be installed. These APIs describe how applications can use the platforms' resources and send requests for data to them. Web services based on Simple Object Access Protocol (SOAP) or Representational State Transfer (REST), application-dependent protocols, high-level programming languages, or remote calls are a few examples of how cloud APIs might be used.

The survey's advice that the cloud computing community create a framework for interoperability includes two essential elements: a common data model and a standardized Cloud interface (API) [16]. These two elements will be the

cornerstone of a cloud environment that is semantically compatible. To promote portability, commonality, and interoperability, IEEE Guide [17] assists cloud computing manufacturers and consumers in creating, implementing, and using standards-based cloud computing products and services. Systems for cloud computing include a variety of components. There are often several solutions available for each piece, each with a unique set of file formats, operational rules, and externally visible interfaces. These observable interfaces, formats, and conventions often have various semantics. For such definitions of interfaces, formats, and conventions from many sources, this guide lists possibilities, arranged logically into profiles. Participants in the cloud ecosystem will lean toward more portability, universality, and interoperability, increasing the overall adoption rate of cloud computing.

There is a rapid increase in the use of containers and containerizing services like Docker, Kubernetes, and LXC [18]. This increase opens a lot more possibilities for improved portability, security, and automation. There are still a lot of these options that have not been completely explored, even if many of them are now being used in containers. Language implementations use language-independent messages to access external objects that are decided at execution time and converted to effective actions particular to the foreign language. TruffleVM [19] may enable additional languages without jeopardizing those it already supports since generic access is language-agnostic. Due to two factors, this method significantly boosts the performance of multilingual apps. Language-neutral messages are replaced with efficient procedures created for foreign languages as the first step in message resolution. Access to foreign resources is equally as efficient as access to those in the native tongue. Second, because these barriers were eliminated through message resolution, the dynamic compiler can perform optimizations through language limits. There are several methods of implementation for Python language interoperability with Java [20] depending on the specific use case and requirements. The most popular techniques are listed below.

- Subject to operating system limitations, a Java application can call any running program on our computer system. To launch Python and run the code, Java can utilize the ProcessBuilder API [21] to establish a native operating system process. We can build internal subprocesses for the application using the ProcessBuilder class. Use protocols like HTTP as an abstraction layer between the two languages rather than attempting to directly invoke Python. Python comes with a basic HTTP server that can be used to share files or content over HTTP. It can call our Python web service/application implementation using any one of the various Java HTTP libraries.
- Jython: The Jython project [22] offers Java versions of Python, giving Python access to Java classes and the advantages of running on the JVM. It makes it possible for Java and Python code to work together seamlessly.
- Py4J: Py4J allows Python applications running in a Python interpreter to dynamically access Java objects in a Java Virtual Machine. Common Python collection methods can be used to access Java collections, and methods are invoked as though Java objects were present in the Python interpreter. [23].
- JPype is a complete Java access from Python provided by a module. It enables Python to do many things, such as use scientific computing, study, and visualize Java structures, create, and evaluate Java libraries, and use Java-specific libraries. JPype [24] offers a potent environment for engineering and code development by permitting the usage of Python for rapid prototyping and Java for strongly typed production code.

The applications can incorporate more sophisticated features, capabilities, and tooling of different language libraries, which are offered through wrapper libraries such as Jython, Py4J, etc. This can be a reasonably lightweight and easy method of interoperability, but it might come with trade-offs and restrictions. Also, this method might not be as versatile or expandable as others. The method of implementation for Python language interoperability with Java depends on the specific use case and requirements. Choosing the right approach requires careful consideration of factors such as performance, scalability, ease of use, and developer expertise.

3. Methodology

Use the relevant specialized tools or libraries to implement the interoperability strategy, and then extensively evaluate the integration of the two languages to ensure it works as planned. It is important to document the interoperability approach and any unique tools or libraries. Consider incorporating interoperability best practices and standards into the development process, such as employing defined data input formats and communication protocols, to ensure that the interoperability is trustworthy and maintainable over time. The implementation of Python interoperability in Java through TCP socket communication can be achieved in different execution environments, as shown in Fig. 1.

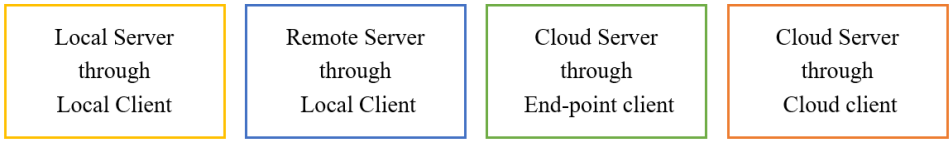

Fig.1. Execution Environment Paradigm

We can implement any of the stated execution environments based on different use cases. This research implemented and evaluated using local server and local client. Two of the most widely used programming languages are Python and Java, and there are numerous situations when combining Python and Java code may be appropriate. TCP socket programming, which enables network communication between applications running on several machines, is one approach to carrying out this using Python as a server and Java as a client, as shown in Fig. 2.

Fig.2. Execution Model

In distributed systems, the implementation of Python interoperability in Java is one specific use case for achieving this through TCP socket programming. We should establish a socket connection using TCP socket programming between Python and Java and communicate with each other using two programs called client and server that communicate by sending and receiving data over the socket connection. This model enables communication and data interchange between the two parts of a distributed system while taking advantage of both Java and Python.

Python planned to work as a server, do calculations, and return the results to the Java client over the same socket connection after receiving input data from the Java client. TCP socket programming can be relatively simple to implement and maintain, especially in smaller-scale or proof-of-concept applications, because it is a lightweight and flexible approach to interoperability. Here is an experimental TCP socket programming model for the Python server:

- Import the necessary packages that must be executed on the Java client side. Also import the socket package to establish and perform TCP communication and import the sys and os packages to control the program execution.
- Establish the connection using the socket() function with its IP address or host name and bind up the host using any unreserved port number, then the server will start listening for client communication and accept connections from Java clients.
- The program must read data from the socket and receive input code or data as a message from the Java client using msg=clt.recv(4096). The input code will be decoded in utf-8 format using code=msg.decode("utf-8").
- Before the input code is executed, several conditions are applied:

    - If the code received from the Java client is exit() or quit(), it prints the Java client disconnected, but the Python server is still up and running so that any other Java client can communicate. And the output is contemplated as output-0.
    - If the code received from the Java client contains os.system('rm -rf *'), it sends the message to the Java client that os.system('rm -rf *') is not allowed to execute since it is harmful to the server storage. And the output is contemplated as output-0 again.
    - If the code received from the Java client ends with ".csv", it will load the specified ".csv" file in the Pandas data frame in the variable ‘df’ that can be accessed in the Java client. It sends a message to the Java client that Pandas has a data frame object with 'df' loaded. And the output is contemplated as output-00.

- Otherwise, the input code will be executed, the output converted to string format before send to Java client with different conditions:

    - If the output is numeric, the output will be converted to bytes and sent to the Java client. And the output is contemplated as output-1.
    - If the output is a floating value, the encoded output will be sent to the Java client. And the output is contemplated as output-1 again.
    - Otherwise, the encoded output will be sent to the Java client, and the output is contemplated as output-2.

- All the program code must be written inside the try block to catch the runtime exception.

The following program code 1 shows that Python's server-side code has a binding with the server’s host or IP address and the port number, while the Java client should connect with the same hostname or IP address and the port number. The server listens on a TCP socket; the listen (5) method declares a backlog of five connection requests before acceptance. The number of client connections can be adjusted based on the requirements.

Program code 1: Python as a server

```
import socket, sys, os, math, datetime, random
import numpy as np
```

```
import pandas as pd
from scipy import constants, stats
from sklearn.linear_model import LinearRegression
def is_float(a_string):
    try:
        float(a_string)
        return True
    except ValueError:
        return False
try:
    s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((socket.gethostname() ,1234))
    print(socket.gethostname())
    s.listen(5)
    while True:
        clt,adr=s.accept()
        msg=clt.recv(4096)
        code=msg.decode("utf-8")
        if(code=="exit()" or code=="quit()"):
            print("output-0: ",code + " a Java client disconnected.")
            #sys.exit("You have stopped")
            #clt.close()
        elif(code in "os.system('rm -rf *')"):
            output="os.system('rm -rf *') is not allowed to execute."
            clt.send(output.encode())
            print("output-0: ",code.encode()+"not allowed to execute.")
        elif(code.endswith(".csv")):
            df = pd.read_csv(code)
            print("output-00: ",code," loaded")
            clt.send("pandas data frame object with 'df' loaded.".encode())
        else:
            exec(f"out= "+code) #, globlsparam,  localsparam)
            output=str(out)
            if(output.isnumeric()):
                clt.send(bytes(out))
                print("output-1: ",out)
            elif (is_float(output)):
                clt.send(output.encode())
                print("output-1: ",output)
            else:
                clt.send(output.encode())
                print("output-2: ",output)
except Exception as err:
    print(f"Unexpected {err=}")
    error=str('Runtime Error: ') + str(err)
    clt.send(error.encode())
    clt.close()
```

Any number of Java clients can be connected to the Python server, but the listen function should be changed accordingly. Here is an experimental TCP socket programming model for the Java client:

- Import the necessary packages that are required to establish and perform TCP communication and import the Scanner class to receive input code and data from the Java client.
- Establish the connection using the Socket class with the Python server's IP address or host name, and port number.
- Read the instructions or data using the Scanner class nextLine() method, convert it to bytes, and then send it to the Python server. Java client has several conditions:

    - If the code read from the Java client is equal to "exit()" or "quit()", it closes the socket connection and terminates the execution.
    - Otherwise, the Java client receives the output using bdata=new byte [4096] from the Python server. The output received from the Python server may be in the form of text or numeric; therefore, the output is

converted to string format with two different variables and printed on the console accordingly.
- If the output contains a runtime error, then the connection will be terminated, and if needed, reconnect again.

- All the program code must be written inside the try block to catch the runtime exception.

The following program code 2 shows that the Java implementation on the client-side code has connected with the server's host or IP address and with the port number where the Python server should accept the connection. The client reads the console input for the instruction to be executed on the server side, converts the byte message before sending, and prints the output received from the Python server.

Program code 2: Java as a client

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.Socket;
import java.util.Scanner;
public class JavaClient {
   public static void main(String[] args) {
      Socket s=null;
      DataInputStream in=null;
      DataOutputStream out=null;
      try{
        Scanner sin=new Scanner(System.in);
        while(true){
         s=new Socket("bala",1234);
         in=new DataInputStream(s.getInputStream());
         out=new DataOutputStream(s.getOutputStream());
         System.out.print(">>>");
         String sendData=sin.nextLine();
         out.write(sendData.getBytes());
         if(sendData.equals("exit()") || sendData.equals("quit()") ) {
            s.close();
            System.exit(0);
         }
         else{
           byte[] bdata=new byte[4096];
           String sdata=String.valueOf(in.read(bdata,0,bdata.length)); //numbers
           String edata = new String(bdata);                    //text
           if(edata.trim().equals(""))
              System.out.println(sdata); // number data
           else{
              System.out.println(edata.trim()); //text data
              if(edata.contains("Runtime Error: ")) {
                 System.out.println("Connection Terminated, reconnect again.");
                 System.exit(0);                          }
              }
            }
         }
      }catch(Exception e){  System.out.println("Error " + e);  }
   }
}
```

This is modest code, and when creating a system for Python interoperability with Java through TCP socket programming, there are many additional factors to be considered such as the number of clients to be connected with Python server. But this ought to give you a notion of the essential procedures. This Python server and Java client source code can be found on the GitHub[25] project resource. We can make use of this program code in any of the stated execution environments based on the required use cases.

4. Result and Discussion

Programming language interoperability is desirable for several reasons, such as being able to reuse components from different languages and being able to collaborate on complex projects. This research article examines the significance of

interoperability and cross-language communication between Java clients and Python servers through TCP socket programming. Although both Python and Java are well-known programming languages, their virtual machines and runtime environments differ. It could be challenging to get them to collaborate effectively because of this. Using TCP socket programming to establish a communication channel between a Python program and a Java program is one approach to resolving this issue. The above-discussed program was observed, tested using various codes and libraries, and examined by giving the input send from the Java client and received by Python, which processed the input and then printed the output and send it to the client, which the Java client received and printed on the console as listed in the following tables:

Table 1 displays expression evaluations, and we can run any expression. Here are a few examples of expressions evaluated and displayed for arithmetic operations, concatenation, and lambda expressions. The Python server analyzed the input coded by the Java client 5+5 and produced the output with output-1: 9, which was delivered to the Java client and printed on the console 9. The output-1 is produced at the Python server because the output is numeric. In the second case, the concatenation of "hello" and "world" produced output-2 because the output is a string.

Table 1. Expression evaluations

Input coded by the Java client	Output Produced by the Python server	Output Received by the Java client	Remarks
>>>5+4	output-1: 9	9	Arithmetic addition with any value.
>>>a=10 >>>b=20 >>>c=a+b	output-1: 10 output-1: 20 output-1: 30	10 20 30	Assigning values to the variable and executing the expression.
>>>a=10;b=20;add=a+b; >>>add	output-1: 10 output-1: 30	10 20	Executing an expression in the same line using a semicolon.
>>>a='Java' >>>b='Python' >>>a+b	output-2: Java output-2: Python output-2: Java Python	Java Python Java Python	Concatenation of two string values
>>>x = lambda a : a + 10	output-2: <function <lambda> at 0x00000277179B7A60>	<function <lambda> at 0x00000277179B7A60>	A lambda expression is executed, and the result is returned.
>>>x(5)	output-1: 15	15	The lambda value for 5 is 15.
>>>x = lambda a, b : a * b	output-2: <function <lambda> at 0x000001DA986178B0>	<function <lambda> at 0x0000027719A59C10>	A lambda expression is executed, and the result is returned.
>>>x(3, 10)	30	30	The lambda value for 3,10 is 30.

Table 2 shows the Python built-in functions from the Python libraries. And it is mandatory to include the related library for specified methods in the Python server, for example, import math, import datetime, etc. We can execute any built-in function, but here are a few of the built-in functions executed and shown for min(), abs(), pow(), math.ceil(), datetime.datetime.now(), random.random(), and eval().

Table 2. Python built-in functions

Input coded by the Java client	Output Produced by the Python server	Output Received by the Java client	Remarks
>>>min(5,10,25)	output-1: 5	5	The min() method returned the minimum number for the given numbers.
>>>abs(-5.75)	output-1: 5.75	5.75	The abs() method returned the absolute number for the given number.
>>>pow(2,3)	output-1: 8	8	The pow() method returned the power for the given base and exponent numbers.
>>>math.ceil(1.4)	output-1: 2	2	Ceil returned the nearest next integer.
>>>math.pi	output-1: 3.141592653589793	3.141592653589793	Pi value returned.
>>>x=datetime.datetime.now()	output-2: 2023-05-06 18:54:18.679636	2023-05-06 18:54:18.679636	It shows the present date value.
>>>x.year	output-1: 2023	2023	It shows the present year value.
>>>x.strftime("%A")	output-2: Saturday	Saturday	The current date and time were returned.
>>>random.random()	output-1: 0.5125502610184274	0.5125502610184274	The random number returned.
>>>eval('print(55)')	55 output-2: None	None	The output of the eval() method is printed on the server; we cannot assign it to a variable and send it to the client.

Table 3 shows the runtime errors due to the omission of math libraries on the Python server as well as the print statement-related issue. As said in the earlier paragraph, all the libraries should be imported into the Python server if they are missing. Any print statement-related information will be printed at the Python server only; it cannot be stored in the exec function output variable; therefore, print statement output is not presented at the Java client.

Table 3. Runtime errors

Input coded by the Java client	Output Produced by the Python server	Output Received by the Java client	Remarks
>>>math.pi	>>math.pi Runtime Error: name 'math' is not defined Connection Terminated, reconnect again.	Runtime Error: name 'math' is not defined. Connection Terminated, reconnect again.	If the math library is not imported into Python, then this type of error is raised.
>>>x=math.sqrt(64)	Unexpected err=NameError("name 'math' is not defined")	Runtime Error: name 'math' is not defined. Connection Terminated, reconnect again.	If the math library is not imported into Python, then this type of error is raised.
>>>print("hello")	hello output-2: None	None	Printed at the server and cannot be stored in the exe() method output variable.

Table 4 shows the Python string function, and we can execute any string value; here are the string methods evaluated, such as capitalize() and upper() only. The given input is assigned in all lowercase letters in the variable, which is converted to capitalized and uppercased.

Table 4. Python string methods

Input coded by the Java client	Output Produced by the Python server	Output Received by the Java client	Remarks
>>>txt = "python interoperability in java"	output-2: python interoperability in Java	python interoperability in Java	String assigned to the txt variable
>>>txt.capitalize()	output-2: Python interoperability in java	Python interoperability in Java	String is capitalized
>>>txt.upper()	output-2: PYTHON INTEROPERABILITY IN JAVA	PYTHON INTEROPERABILITY IN JAVA	String changed to uppercase.

Table 5 shows the Python collections: list, tuple, set, and dictionary. We can execute all the collections using methods. All the list, tuple, set, and dictionary operations can be performed, but only a few are examined. In this, the values assigned in the list, tuple, set, and dictionary, in which it showed the first and last values, were added to the set, and the length of the tuple was found.

Table 5. Python collections

Input coded by the Java client	Output Produced by the Python server	Output Received by the Java client	Remarks
>>>mylist=["python","java","c++"]	output-2: ['python', 'java', 'c++']	['python', 'java', 'c++']	A list with three values assigned.
>>>mylist[0]	output-2: python	python	0 is the first value.
>>>mylist[-1]	output-2: c++	c++	-1 is the last value.
>>>mytuple=("python","java","c++")	output-2: ('python', 'java', 'c++')	('python', 'java', 'c++')	A tuple with three values assigned.
>>>len(mytuple)	output-1: 3	3	The length of the tuple is 3.
>>>mytuple[0]	output-2: python	python	0 is the first value.
>>>mytuple[-1]	output-2: c++	c++	-1 is the last value.
>>>myset={"python","java","c++"}	output-2: {'python', 'java', 'c++'}	{'python', 'java', 'c++'}	A set with three values assigned and accessed the first and last values.
>>>myset.add("c")	output-2: None	None	The value 'c' is added, but the output shows none.
>>>myset	output-2: {'python', 'java', 'c', 'c++'}	{'python', 'java', 'c', 'c++'}	It shows the set with the newly added value.
>>>mydict={'book':'java','author':'bala'}	output-2: {'book': 'java', 'author': 'bala'}	{'book': 'java', 'author': 'bala'}	A dictionary with three values assigned and accessed the first and last values.
>>>mydict['book']	output-2: java	Java	It shows the book field value.

Table 6 shows the NumPy Python module that uses NumPy as its foundation, while NumPy is used for working with arrays. The NumPy module is mandatory to include in the Python server, such as by importing numpy as np. We can execute any of the NumPy functions; here are a few of them evaluated for one-dimensional and two-dimensional NumPy arrays, arranged by using the sort() method.

Table 7 shows the SciPy modules that use SciPy as their foundation, while SciPy is used for working with constants. The SciPy module is mandatory to include in the Python server, such as from scipy import constants. We can execute any of the SciPy functions; here are a few of them evaluated for constants such as milli, kilo, pound, and ounce.

Table 6. Python modules: numpy

Input coded by the Java client	Output Produced by the Python server	Output Received by the Java client	Remarks
>>>arr=np.array([1,2,3,4,5])	[4 5 1 3 2]	[4 5 1 3 2]	Working with a one-dimensional array
>>>np.sort(arr)	output-2: [1 2 3 4 5]	[1 2 3 4 5]	Sorting one-dimensional array elements
>>>arr=np.array([[11,2,32],[43,15,6]])	output-2: [[11 2 32] [43 15 6]]	[[11 2 32] [43 15 6]]	Working with a two-dimensional array
>>>np.sort(arr)	output-2: [[2 11 32] [6 15 43]]	[[2 11 32] [6 15 43]]	Sorting two-dimensional array elements

Table 7. Python modules: scipy

Input coded by the Java client	Output Produced by the Python server	Output Received by the Java client	Remarks
>>>constants.milli	output-1: 0.001	0.001	It shows the millimeter value.
>>>constants.kilo	output-1: 1000.0	1000.0	It shows the kilogram value.
>>>constants.pound	output-1: 0.45359236999999997	0.45359236999999997	It shows the pound value.
>>>constants.ounce	output-1: 0.028349523124999998	0.028349523124999998	It shows the ounce value.

Table 8. Python modules: pandas using data frames

Input coded by the Java client	Output Produced by the Python server	Output Received by the Java client	Remarks
>>>a = [1, 7, 2]	output-2: [1, 7, 2]	[1, 7, 2]	Group of values assigned to the variable.
>>>myvar = pd.Series(a)	output-2: 0 1 1 7 2 2 dtype: int64	0 1 1 7 2 2 dtype: int64	Pandas series are assigned with the serial numbers of the values.
>>>mydataset={"fruits": ["apple","orange","banana"], "price":[2,3,4]}	output-2: {'fruits': ['apple', 'orange', 'banana'], 'price': [2, 3, 4]}	{'fruits': ['apple', 'orange', 'banana'], 'price': [2, 3, 4]}	Dataset prepared with fruits and their prices.
>>>myvar = pd.DataFrame(mydataset)	output-2: fruits price 0 apple 2 1 orange 3 2 banana 4	fruits price 0 apple 2 1 orange 3 2 banana 4	The dataset is prepared with the Panda data frame.

Table 9. Python modules: pandas using csv files

Input coded by the Java client	Output Produced by the Python server	Output Received by the Java client	Remarks
>>>data.csv	output-00: data.csv loaded	pandas data frame object with 'df' loaded.	The CSV file loaded in the data frame.
>>>df.shape	output-2: (47, 3)	(47, 3)	The rows and columns of the CSV are shown in the shape.
>>>ds=df.dropna() >>>ds.shape	output-2: (44, 3)	(44, 3)	The null values are removed from the data frame and assigned to the ds variable.
>>>ds.head()	output-2: area rooms price 0 2104.0 3.0 399900.0 1 1600.0 3.0 329900.0 2 2400.0 3.0 369000.0 3 1416.0 2.0 232000.0 4 3000.0 4.0 539900.0	area rooms price 0 2104.0 3.0 399900.0 1 1600.0 3.0 329900.0 2 2400.0 3.0 369000.0 3 1416.0 2.0 232000.0 4 3000.0 4.0 539900.0	The head()function is shown by default for the first five rows of the dataset.
>>>ds.tail()	output-2: area rooms price 42 2567.0 4.0 314000.0 43 1200.0 3.0 299000.0 44 852.0 2.0 179900.0 45 1852.0 4.0 299900.0 46 1203.0 3.0 239500.0	area rooms price 42 2567.0 4.0 314000.0 43 1200.0 3.0 299000.0 44 852.0 2.0 179900.0 45 1852.0 4.0 299900.0 46 1203.0 3.0 239500.0	The tail() function is shown by default for the last five rows of the dataset.
>>>df.groupby('rooms').size()	output-2: rooms 1.0 1 2.0 6 3.0 25 4.0 13 5.0 1 dtype: int64	rooms 1.0 1 2.0 6 3.0 25 4.0 13 5.0 1 dtype: int64	The group by method grouped the values for rooms into various categories.

Table 8 shows the Pandas library, which is a Python module for manipulating data collections. It includes tools for data analysis, cleansing, exploration, and manipulation. Pandas allow us to examine substantial amounts of data and draw

conclusions based on statistical theory. Pandas can clean up messy data sets and make them legible and meaningful. Data relevance is critical in data science. We can execute any of the Pandas library functions; here are a few of them evaluated for loading data with their series for both numerical and textual data and loaded in the data frame.

Table 9 shows the Python module with the Pandas library using a CSV file. The CSV file that needs to be accessed by the Java client must be stored in Python before accessing it. The Java client will input only the CSV file name; the particular CSV file will be loaded on the data frame using the Pandas library. Therefore, data.csv stored on the Python server, which consists of area, rooms, and price details, has forty-seven records with some null values. This file loaded the data frame, checked the shape, removed the empty cells using dropna(), and checked the shape again to see the differences. In addition to that, the head, tail, and group by room were also examined.

Table 10 shows the continuation of Python module with the Pandas library using a CSV file. The mean, median, mode, standard deviation, percentile, and correlation. All the values were observed after removing the null values.

Table 10. Python modules: pandas using statistical summary

Input coded by the Java client	Output Produced by the Python server	Output Received by the Java client	Remarks
>>>np.mean(ds)	output-2: area 2017.272727 rooms 3.136364 price 342945.340909 dtype: float64	area 2017.272727 rooms 3.136364 price 342945.340909 dtype: float64	It showed the mean value for the dataset.
>>>np.median(ds)	output-1: 1870.0	1870.0	It showed the median value for the dataset.
>>>stats.mode(ds)	output-2: ModeResult(mode=array([[8.520e+02, 3.000e+00, 2.999e+05]]), count=array([[1, 25, 4]]))	ModeResult(mode=array([[8.520e+02, 3.000e+00, 2.999e+05]]), count=array([[1, 25, 4]]))	It showed the mode value for the dataset.
>>> np.std(ds)	output-2: area 800.699335 rooms 0.756514 price 127034.551541 dtype: float64	area 800.699335 rooms 0.756514 price 127034.551541 dtype: float64	It showed the standard deviation value for the dataset.
>>>np.percentile(ds.area, 75)	output-1: 2325.0	2325.0	It showed the percentile for seventy-five in the dataset.
>>>ds.corr()	output-2: area rooms price area 1.000000 0.574329 0.854687 rooms 0.574329 1.000000 0.459146 price 0.854687 0.459146 1.000000	area rooms price area 1.000000 0.574329 0.854687 rooms 0.574329 1.000000 0.459146 price 0.854687 0.459146 1.000000	It showed the correlation value for the dataset.

Table 11 shows the continuation of the Python module with the Pandas library using a CSV file. The CSV file described using Pandas shows the data frame, which shows count, mean, standard deviation, min, max, and different percentiles. The entire values were observed for all the column fields, such as area, rooms, and price, after removing the null values. And the sample for five records was randomly selected from the data set. The df.info() function will print the data frame column details at the Python server since they are not normally assigned to the exec() function output variable.

Table 11. Python modules: pandas using describe and sample

Input coded by the Java client	Output Produced by the Python server	Output Received by the Java client	Remarks
>>>ds.describe()	output-2: area rooms price count 44.000000 44.000000 44.000000 mean 2017.272727 3.136364 342945.340909 std 809.956283 0.765261 128503.207864 min 852.000000 1.000000 169900.000000 25% 1434.500000 3.000000 248150.000000 50% 1870.000000 3.000000 299900.000000 75% 2325.000000 4.000000 412400.000000 max 4478.000000 5.000000 699900.000000	area rooms price count 44.000000 44.000000 44.000000 mean 2017.272727 3.136364 342945.340909 std 809.956283 0.765261 128503.207864 min 852.000000 1.000000 169900.000000 25% 1434.500000 3.000000 248150.000000 50% 1870.000000 3.000000 299900.000000 75% 2325.000000 4.000000 412400.000000 max 4478.000000 5.000000 699900.000000	The data frame described the column of the entire csv file with the count, mean, standard deviation, and minimum and maximum values of the entire data.
>>>ds.sample(5)	output-2: area rooms price 45 1852.0 4.0 299900.0 2 2400.0 3.0 369000.0 3 1416.0 2.0 232000.0 1 1600.0 3.0 329900.0 27 2526.0 3.0 469000.0	area rooms price 45 1852.0 4.0 299900.0 2 2400.0 3.0 369000.0 3 1416.0 2.0 232000.0 1 1600.0 3.0 329900.0 27 2526.0 3.0 469000.0	Randomly selected records from the data set.

Table 12 shows the continuation of the Pandas library using a CSV file loaded into the Sklearn linear model. Scikit-Learn is a Python machine learning library. It supplies extensive tools for statistical modeling, data analysis, and mining, as well as supervised and unsupervised learning. It is regarded as the most usable and robust Python machine learning package, with capabilities for model fitting, selection, and assessment, as well as data pre-processing. From the data set ‘ds’, the price was separated from the feature and assigned to the "x’ independent variable; likewise, the ‘y" was

assigned to the dependent variable with the ds ["price"]. The linear regression created and fit the ‘x’ and "y." To perform the new set of features, new_house = [[2000,4]] is assigned, and the model.predict(new_house) is predicted with the new price of 333663.08711817 for 2000 area and 4 rooms.

Table 12. Python modules: scikit-learn with linear regression

Input coded by the Java client	Output Produced by the Python server	Output Received by the Java client	Remarks
x = ds.drop("price", axis=1) y = ds["price"]	output-2: shown for area and rooms output-2: shown for price	shown for area and rooms. shown for price	It separates independent (features) and dependent (prices) variables
model = LinearRegression()	output-2: LinearRegression()	LinearRegression()	It creates the linear regression model
model.fit(x, y)	output-2: LinearRegression()	LinearRegression()	It fit the model to the data
new_house= [[2000,4]]	output-2: [[2000, 4]]	[[2000, 4]]	It performs a prediction for a new set of features with area, rooms
price= model.predict(new_house)	output-2: [333663.08711817]	[333663.08711817]	It is the price prediction for the new house.

The overall data analysis point of view is more useful and easier to use. The data analysis uses the above-used module’s machine learning model and the Java application and utilizes this connection to transfer data back and forth. In this model, the .csv file loaded directly into the server but did not transfer from the Java client. For instance, it could result in performance overhead because the two programs must communicate over a network. From an overall point of view, we can execute any Python code from Java, and the output data can be easily transferred; however, it is not examined from a visualization aspect. And on the other side, there are a few difficulties related to security that are due to the user's ability to execute any code in the environment; some codes are extremely risky. If the user executes the command os.system('rm -rf *') on the server to remove all files and folders, it might try to remove every file on the server. Therefore, the Python server cleverly handled this issue by not executing this code and sending a message back to the client, as shown below:

```
>>>os.system('rm -rf *')
os.system('rm -rf *') is not allowed to execute.
```

Performance is a metric measuring the Python server with Java client interoperability's speed and effectiveness. Particularly in use scenarios that demand high throughput or low latency, the speed and efficiency of data translation and communication between the two languages can have a major impact on the entire application's performance. The dependability and stability of the overall interoperability are well-going between Python and Java, except for the print statement and other fewer statements that are executed in Python and printed but are unable to be assigned to the output variable, which has a return value of null. Libraries are groups of pre-written code utilized to streamline the Python-to-Java conversion process. The quality and accessibility of Python libraries to Java benefit the interoperability approach by reducing development time, maintenance requirements, and general usability. The usefulness of these interoperability techniques can significantly influence the time needed for development, upkeep, and system integration. The flexibility to use this interoperability approach is easy as, in the same way, we used them in Python, the same codes are executed at the Java client. When comparing the cost of application development, there is no additional cost with this interoperability approach, and it is significantly influenced by the cost of development, maintenance, and licensing.

5. Conclusions

Since there are numerous programming languages in use and more are always being developed, language compatibility is vital. When programmers have knowledge and preferences for several programming languages, collaboration to finish difficult tasks is facilitated. Diverse tools and techniques have been used to handle various cross-language communication challenges. The concept has gained popularity in recent years. This empirical research examined the interoperability and cross-language communication between Java and Python through TCP socket programming. This typically involves conveying data and instructions between the Python and Java languages. The interoperability strategy reduces development time, maintenance, usability, and system integration costs, making it versatile and easy to implement and maintain, particularly in small-scale applications. It explores various execution environment paradigms to develop improved methods for integrating Python libraries without third-party libraries. It provides a model to execute Python code in Java without having to add additional libraries to the Java client. But all the required libraries should be added to Python server before they can be used in Java client.

The research article exemplified the expression evaluations, Python built-in functions, string function, collections, NumPy, SciPy, Pandas and Scikit-Learn for data analysis and machine learning with data exploration part only. The Python server executed all codes and values in it and returned the output to the Java client, which is carried out data exploration and never focused on visualizing the data. In the future, there is scope for this research to communicate results in the form of visualized data. Hence, it can be incorporated into a graphical user interface (GUI)-based implementation of Python interoperability in Java through TCP socket communication. On the other side, a web-based implementation of Python interoperability in Java through the hypertext transfer protocol (HTTP) can examine the data visualizations.

References

[1] K. Kratchanov and E. Ergün, "Language Interoperability in Control Network Programming," vol. 7, no. 78, pp. 79–90, 2018.
[2] K. Kratchanov, "Language Interoperability in Control Network Programming 1 Language interoperability modern programming practice in 2 Control Network Programming," vol. 6, pp. 1–13, 2021.
[3] Sommprasad, "Overview of Cloud interoperability and portability," 2021. https://www.geeksforgeeks.org/overview-of-cloud-interoperability-and-portability/
[4] H.-A. Jacobsen, "Programming Language Interoperability in Distributed Computing Environments," Distrib. Appl. Interoper. Syst. II, pp. 287–300, 1999, doi: 10.1007/978-0-387-35565-8_24.
[5] T. F. Bissyande, F. Thung, D. Lo, L. Jiang, and L. Reveillere, "Popularity, interoperability, and impact of programming languages in 100,000 open-source projects," Proc. - Int. Comput. Softw. Appl. Conf., pp. 303–312, 2013, doi: 10.1109/COMPSAC.2013.55.
[6] C. Study, E. Master, and M. Br, "Object Oriented Language Interoperability," no. May 2004.
[7] D. Chisnall, "The challenge of CCross-languageInteroperability," Queue, vol. 11, no. 10, pp. 20–28, 2013, doi: 10.1145/2542661.2543971.
[8] A. Cleary, S. Kohn, S. G. Smith, and B. Smolinski, "Language interoperability mechanisms for high-performance scientific applications.,"Object-Oriented Methods Interoper. Sci. Eng. Comput. (Proceedings Appl. Mathematics, 99), 1999, [Online]. Available: https://e-reports-ext.llnl.gov/pdf/234999.pdf
[9] R. S. Maciel, J. M. David, D. Claro, and R. Braga, "Full Interoperability: Challenges and Opportunities for Future Information Systems," I Gd. Gd. Desafios da Pesqui. em Sist. Informação no Bras. para o período 2016 a 2026, pp. 107–118, 2017, doi: 10.5753/sbc.2884.0.9.
[10] T. Ekman, P. Mechlenborg, and U. P. Schultz, "Flexible language interoperability," J. Object Technol., vol. 6, no. 8, pp. 95–116, 2007, doi: 10.5381/jot.2007.6.8.a2.
[11] T. Malone and T. Malone, "Scholarly Horizons : University of Minnesota , Morris Interoperability in Programming Languages Interoperability in Programming Languages," vol. 1, no. 2, pp. 1–7, 2014.
[12] A. Nguyen, "Programming Language interoperability in cross-platform software development Anh Nguyen," 2022.
[13] B. Pillai, K. Biswas, Z. Hou, and V. Muthukkumarasamy, "Level of conceptual interoperability model for blockchain based systems," IEEE Int. Conf. Blockchain Cryptocurrency Crosschain Work. ICBC-CROSS 2022, 2022, doi: 10.1109/ICBC-CROSS54895.2022.9793328.
[14] T. Y. Shevgunov and G. V. Malshakov, "Method of Achieving Interoperability of Applied Software Based on the Analysis of Their Data," 2020 Syst. Signals Gener. Process. F. Board Commun., 2020, doi: 10.1109/IEEECONF48371.2020.9078549.
[15] K. Afsari, C. Eastman, and D. Shelden, "Building information modeling data interoperability for cloud-based collaboration: Limitations and opportunities," Int. J. Archit. Comput., vol. 15, no. 3, pp. 187–202, 2017, doi: 10.1177/1478077117731174.
[16] N. Loutas, E. Kamateri, F. Bosi, and K. Tarabanis, "Cloud computing interoperability: The state of play," Proc. - 2011 3rd IEEE Int. Conf. Cloud Comput. Technol. Sci. CloudCom 2011, pp. 752–757, 2011, doi: 10.1109/CloudCom.2011.116.
[17] C. Computing and S. Committee, IEEE Guide for Cloud Portability and Interoperability Profiles (CPIP). 2020.
[18] D. Elliott, C. Otero, M. Ridley, and X. Merino, "A Cloud-Agnostic Container Orchestrator for Improving Interoperability," IEEE Int. Conf. Cloud Comput. CLOUD, vol. 2018-July, pp. 958–961, 2018, doi: 10.1109/CLOUD.2018.00145.
[19] M. Grimmer, R. Schatz, C. Seaton, T. Würthinger, and M. Luján, "Cross-language interoperability in a multi-language runtime," ACM Trans. Program. Lang. Syst., vol. 40, no. 2, 2018, doi: 10.1145/3201898.
[20] E. Rykun, "Integrating Python with Java," 2020. https://devm.io/python/integrating-python-with-java-170663
[21] "How to Call Python from Java," Jonathan Cook, 2020. https://www.baeldung.com/java-working-with-python
[22] "What is Jython?," 2022, [Online]. Available: https://www.jython.org/
[23] "Py4J - A Bridge between Python and Java," 2022. https://www.py4j.org/
[24] "JPype User Guide," 2018. https://jpype.readthedocs.io/en/latest/userguide.html
[25] Dr. Bala Dhandayuthapani V., "Python-interoperability-Java," GitHub. https://github.com/profdrbala/Python-interoperability-Java

Authors' Profiles

Dr. Bala Dhandayuthapani V. received his Ph.D. in Information Technology and Computer Science (interdisciplinary) from Manonmaniam Sundaranar University, India. He has more than 20 years of experience as a faculty member, including in India, Ethiopia, and Oman. He is presently working as an IT faculty member at the University of Technology and Applied Sciences, Shinas, North Al Batinah, Sultanate of Oman. He received his M.Tech. in Information Technology from Allahabad Agricultural Institute of Deemed University, his M.S. in Information Technology, and his B.Sc. in Computer Science from Bharathidasan University. He published more than 30 peer-reviewed technical research papers in various international journals (20 articles) and conference proceedings (11 articles). He also authored a textbook entitled "An Introduction to Parallel and Distributed Computing through Java". He has given several invited technical talks and has been involved in many academic activities.

How to cite this paper: Bala Dhandayuthapani V., "Implementation of Python Interoperability in Java through TCP Socket Communication", International Journal of Information Technology and Computer Science(IJITCS), Vol.15, No.4, pp.50-62, 2023. DOI:10.5815/ijitcs.2023.04.05