Woche 8: Netzwerke mit Python

Einleitung

Die folgenden drei Wochen unseres Trainings sind einer spannenden und zunehmend wichtigen Disziplin in der Welt der Programmierung gewidmet: der Netzwerkprogrammierung mit Python.

Denn in unserer zunehmend vernetzten Welt ist das Verständnis, wie Daten über Netzwerke übertragen werden, von zentraler Bedeutung. Ob Sie eine Karriere in der System- und Netzwerkadministration, in der Cybersicherheit oder in der Softwareentwicklung anstreben – diese Fähigkeiten sind unerlässlich in einer Vielzahl von Anwendungsbereichen – von der Entwicklung moderner Webanwendungen über die Sicherstellung der Netzwerksicherheit bis hin zur Automatisierung von Systemadministration.

In dieser Woche beginnen wir mit den Grundlagen. Sie werden die Grundprinzipien der Netzwerkkommunikation kennenlernen, einschließlich der Funktionsweise verschiedener Netzwerkprotokolle wie TCP/IP und UDP. Wir werden untersuchen, wie Python für die Erstellung einfacher Netzwerkanwendungen genutzt werden kann, und Sie werden lernen, wie man grundlegende Client- und Server-Anwendungen in Python erstellt. Diese Woche ist darauf ausgerichtet, Ihnen ein festes Fundament in der Netzwerkprogrammierung zu geben, auf dem die nächsten Wochen aufbauen werden.

In dieser ersten Woche zum Thema Netzwerkprogrammierung mit Python, behandeln wir folgende Themen:

- Einführung in Netzwerkprotokolle und Kommunikationsmodelle.
- Praktische Erfahrungen mit der Erstellung von Client-Server-Applikationen in Python.
- Einblicke in die Handhabung von Netzwerkdaten und die Bearbeitung von Netzwerkfehlern.
- Projekte und Übungen, die Ihr Wissen festigen und erweitern.

Bereiten Sie sich darauf vor, in die Welt der Netzwerkprogrammierung einzutauchen und Fähigkeiten zu entwickeln, die Sie in die Lage versetzen, leistungsfähige und effiziente Netzwerkanwendungen zu erstellen.

Gesamtüberblick

Hier ein Überblick über die Inhalte und Aktivitäten der aktuellen Woche:

- Selbststudium:
    - Einführung in die Netzwerkkommunikation und Verständnis grundlegender Netzwerkprotokolle wie TCP/IP und UDP.
    - Untersuchung des Client-Server-Modells und dessen Funktionsweise.
    - Erstellen und Testen von einfachen Client- und Server-Anwendungen in Python.
    - Verstehen und Handhaben von Netzwerk-Sockets und deren Kommunikationsabläufen.
    - Methoden der Datenübertragung und -empfang über Netzwerke.
    - Verarbeitung und Manipulation von Netzwerkdaten in Python.
- Aufgaben:
    - Entwicklung einer einfachen Client-Server-Anwendung, die Basisfunktionen wie das Senden und Empfangen von Nachrichten umfasst.
    - Experimentieren mit verschiedenen Arten von Netzwerkkommunikation und dem Umgang mit Netzwerkfehlern.
- Präsenztag:
    - Wiederholung
    - Vertiefung: Websockets
    - Vertiefung: Protokolle
    - Vertiefung: Client-Server-Anwendungen
    - Erweiterte Themen: asynchrone Kommunikation.
    - Ausblick

Inhalte und Thematisch Abgrenzung

Die folgende Auflistung zeigt detailliert, welche Themen Sie in der Woche behandeln und bearbeiten. Sie sind eine Voraussetzung für die folgenden Wochen und sollten gut verstanden worden sein. Wenn es Verständnisprobleme gibt, machen Sie sich Notizen und fragen Sie am Präsenztag nach, so dass wir gemeinsam zu Lösungen kommen können. Und denken Sie bitte immer daran: es gibt keine „dummen“ Fragen!

1. Einführung in die Netzwerkprogrammierung:
   - Grundlagen und Konzepte der Netzwerkprogrammierung.
   - Verständnis von Netzwerkprotokollen wie TCP/IP und UDP.
2. Erstellen von Client- und Server-Anwendungen in Python:
   - Grundlagen der Socket-Programmierung.
   - Erstellen einfacher Client- und Server-Anwendungen mit Sockets in Python.
3. Datenübertragung über Netzwerke:
   - Methoden zum Senden und Empfangen von Daten über Netzwerke.
   - Handhabung von Datenströmen und Nachrichten in Netzwerkanwendungen.
4. Netzwerkfehler und Ausnahmen:
   - Erkennen und Behandeln von Netzwerkfehlern in Python.
   - Implementierung robuster Fehlerbehandlungsmechanismen.
5. Aspekte der Netzwerksicherheit:
   - Einführung in die grundlegenden Konzepte der Netzwerksicherheit.
   - Verwendung sicherer Protokolle und Techniken in der Netzwerkprogrammierung.
6. Multithreading in Netzwerkanwendungen:
   - Einführung in Multithreading und dessen Anwendung in der Netzwerkprogrammierung.
   - Erstellen von Serveranwendungen, die mehrere Client-Verbindungen gleichzeitig verwalten können.
7. Anwendung der gelernten Konzepte in einem umfangreichen Netzwerkprojekt.

Lernpfad

Der Lernpfad ist ein Vorschlag, in welcher Reihenfolge Sie die Inhalte der Woche angehen können. Betrachten Sie ihn gerne als eine Todo-Liste, die Sie von oben nach unten abhaken. So können Sie sicher sein, dass Sie alle wichtigen Themen bearbeitet haben und sind gut vorbereitet für die folgenden Wochen.

1. Einführung in die Netzwerkkommunikation:
   - Beginnen Sie mit den Grundlagen der Netzwerkkommunikation und den zentralen Netzwerkprotokollen wie TCP/IP und UDP.
   - Untersuchen Sie die Funktionen und Eigenschaften des Client-Server-Modells.
2. Erstellen einfacher Netzwerkanwendungen:
   - Lernen Sie, wie man grundlegende Client- und Server-Anwendungen in Python erstellt.
   - Experimentieren Sie mit der Erstellung und dem Testen von Netzwerk-Sockets.
3. Datenübertragung und -verarbeitung:
   - Erforschen Sie Methoden zur Übertragung von Daten über Sockets und deren Verarbeitung in Python.
   - Üben Sie das Senden und Empfangen von Nachrichten zwischen einem Client und einem Server.
4. Netzwerkprotokolle in der Praxis:
   - Vertiefen Sie Ihr Verständnis für TCP und UDP durch praktische Anwendungsfälle.
   - Erstellen Sie einfache Anwendungen, die diese Protokolle nutzen.
5. Fehlerbehandlung in der Netzwerkprogrammierung:
   - Lernen Sie, wie Sie mit Netzwerkfehlern und Ausnahmen umgehen.
   - Üben Sie die Implementierung robuster Fehlerbehandlungsmechanismen in Ihren Netzwerkanwendungen.
6. Projektaufgabe: Client-Server-Kommunikation:
   - Entwickeln Sie ein Projekt, das eine Client-Server-Kommunikation umfasst, wie z.B. einen einfachen Nachrichtenaustausch oder Datentransfer.
   - Testen und optimieren Sie Ihre Anwendung für Effizienz und Zuverlässigkeit.

Programmieraufgaben

Die folgenden Programmieraufgaben sollen Ihnen eine Anregung geben. Haben Sie eigene Ideen und Themen, die Sie ausprobieren wollen, dann sollten Sie diesen nachgehen. Wichtig ist vor allem, dass Sie „Dinge ausprobieren“. Und auch, dass Sie Fehler machen, sowohl syntaktische als auch semantische. Versuchen Sie diese Fehler zu finden und aufzulösen, dann gerade aus den Fehlern lernen Sie am Ende am meisten.

1. Erstellen eines einfachen Echo-Servers:

   Entwickeln Sie einen einfachen Server, der empfangene Nachrichten an den Client zurücksendet (Echo). Verwenden Sie Sockets in Python, um einen Server zu erstellen, der Verbindungen von Clients akzeptiert und die empfangenen Nachrichten zurücksendet.

2. Entwicklung eines Chat-Clients:

   Erstellen Sie einen einfachen Chat-Client, der es ermöglicht, Nachrichten an einen Server zu senden und Antworten zu empfangen. Implementieren Sie einen Client, der sich mit Ihrem Echo-Server verbindet und eine bidirektionale Kommunikation ermöglicht.

3. Dateiübertragung zwischen Client und Server:

   Entwickeln Sie eine Anwendung, die das Hoch- und Herunterladen von Dateien zwischen einem Client und einem Server ermöglicht. Implementieren Sie sowohl Server- als auch Client-Seite, die es erlauben, Dateien in beide Richtungen zu übertragen.

Abschluss-Quiz

Das Quiz soll Ihnen einen ersten Hinweis auf Ihren Lernfortschritt geben. Nach unserer Einschätzung sollten Sie diese Fragen alle beantworten können, wenn Sie den Stoff der Woche durchgearbeitet und verstanden haben. Natürlich gibt es noch sehr viel mehr mögliche Fragen, dazu wollen wir auf die Literatur und das Internet verweisen. Geben Sie gerne einmal „python quizzes“ bei Google ein.

1. Wie erstellt man in Python einen Socket für eine TCP-Verbindung?

   a) socket.socket(socket.AF_INET, socket.SOCK_STREAM)

   b) socket.tcp()

   c) tcp.socket()

   d) socket.create_tcp()

2. Welche Methode wird in Python verwendet, um auf einem Socket zu lauschen (listen) und Verbindungen zu akzeptieren?

a) socket.accept()

b) socket.listen()

c) socket.bind()

d) socket.connect()

3. Welche Python-Bibliothek wird typischerweise für die Implementierung von HTTP-Anfragen verwendet?

a) requests

b) http

c) socket

d) urllib

4. In der Netzwerkprogrammierung mit Python, wofür wird socket.bind() verwendet?

a) Zum Senden von Daten über einen Socket.

b) Zum Verbinden eines Sockets mit einer Netzwerkadresse.

c) Zum Anhören eingehender Anfragen auf einem bestimmten Port.

d) Zum Akzeptieren einer Verbindung.

5. Was ist der Zweck der send() und recv() Methoden in der socket-Bibliothek in Python?

a) send() zum Senden und recv() zum Empfangen von Daten über einen Socket.

b) send() zum Verbinden und recv() zum Trennen von Sockets.

c) send() zum Abfragen und recv() zum Empfangen von HTTP-Anfragen.

d) send() und recv() werden in Python nicht verwendet.

6. Wie erzeugt man in Python einen UDP-Socket?

a) socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

b) socket.udp()

c) udp.socket()

d) socket.create_udp()

7. Was ist eine gängige Anwendung für das select Modul in der Python-Netzwerkprogrammierung?

a) Zum Parsen von HTML.

b) Für die gleichzeitige Überwachung mehrerer Sockets.

c) Zum Verschlüsseln von Daten.

d) Zur Verbesserung der Datenübertragungsgeschwindigkeit.

8. Wie wird in Python eine Verbindung von einem Client zu einem Server-Socket hergestellt?

a) socket.connect()

b) socket.bind()

c) socket.listen()

d) socket.accept()

9. Welche Aussage über das socket-Modul in Python ist korrekt?

a) Es unterstützt nur die TCP-Netzwerkprotokolle.

b) Es wird hauptsächlich für die Entwicklung von GUI-Anwendungen verwendet.

c) Es bietet Low-Level-Netzwerkfunktionen.

d) Es ist ein High-Level-HTTP-Client.

10. Was ist der Hauptunterschied zwischen socket.send() und socket.sendall() in Python?

a) send() sendet einmal, sendall() wiederholt das Senden bis alle Daten übertragen sind.

b) send() ist für UDP, sendall() für TCP.

c) Es gibt keinen Unterschied, beide Funktionen sind identisch.

d) send() benötigt eine Netzwerkadresse, sendall() nicht.

Ressourcen

Hier nun die Verweise auf Lernquellen, die uns für diese Woche und ihre Inhalte geeignet erscheinen. Je nachdem, welcher Lerntyp Sie sind, wählen Sie sich ihre bevorzugte Quelle, es ist nicht zwingend notwendig alle durchgearbeitet zu haben. Allerdings sollten die Inhalte des Lernpfads angesprochen und verstanden worden sein.

- **Buch**: Introduction to Python Network Automation

  Paramiko und Netmiko

  CISCO Router

- **Buch**: Network Programming in Python: The Basics

  Umfangreiche Einführung in LowLevel Funktionalität im Netz

  Alle wichtigen Protokolle

- **Video**s:
  - Python Web Requests
  - Building Restful Web Services with Python
- **Lab and Course**:
  - Lab: Building Restful Web Services with Python
  - Course: Learn Web Scraping with Beautiful Soup