Berufsakademie Mosbach
- Staatliche Studienakademie -
Lohrtalweg 10

74821 Mosbach

Skriptsprachen am Beispiel von Python

Firma / Anschrift	dsb AG Konrad-Zuse-Straße 16 74172 Neckarsulm
Fachrichtung	Wirtschaftsinformatik
Name	Timo Steidle
Anschrift	Pappelweg 5 74177 Bad Friedrichshall
Abgabedatum	

Die Praxisarbeit ist formal und inhaltlich geprüft und zur Einreichung an die Berufsakademie freigegeben!

Unterschrift Ausbildungsbeauftragter

Stempel des Ausbildungsbetriebs

Inhaltsverzeichnis

„Der fundamentale Akt von Freundschaft unter denkenden Wesen besteht darin, einander etwas beizubringen und Wissen gemeinsam zu nutzen.“ [1]

1 Richard Stallman, Free Software Foundation (FSF)

1. Einführung

1.1 Vorstellung des Ausbildungsbetriebs dsb AG [2]

Die dsb AG wurde 1966 als Daten - Service Beck GmbH & Co gegründet. Das Ziel war eine Verbesserung des Zeitschriftenvertriebes für Medienunternehmen. Heute ist die dsb AG ein Application Service Provider in den Bereichen Abonnementmanagement (AVS), Zeitschriftenvertrieb (ZHS) und Versandhandel (VHS).

Die rund 150 Mitarbeiter der Firma erstellen maßgeschneiderte Branchenlösungen für ihre Kunden. Dabei werden sie von dem Tochterunternehmen dsb München AG und der Niederlassung in Hamburg durch deren Marketing -, Vertriebs - und IT – KnowHow unterstützt.

Bei dem Geschäftsmodell des Application Service Provider, kurz ASP, lagert der Kunde seine benötigten EDV – Ressourcen samt Hard- und Software aus und greift via Internet oder Direktleitung auf seine Ressourcen bei einem ASP zurück.

1.2 Die Abteilung „dezentrale Systeme“

Die dsb AG unterhält ungefähr 90 aktive Server, unzählige VPN – Tunnel und Netzwerke. Die Administration und Pflege der Systeme wird in der Abteilung „dezentrale Systeme“ durchgeführt. Der Aufgabenbereich der Abteilung ist weit gestreut, beginnend von der Wartung einzelner Workstations über das Einrichten von VPN - Tunnel bis hin zur Installation und Produktivsetzung von Server. Ein wichtiger Bestandteil der Arbeit ist auch das Auswerten diverser Log – Dateien, mit deren Hilfe man Fehler nachvollziehen und zurückverfolgen kann und mit denen man den Überblick über die Netzwerke und Verbindungen behält.

Dabei ist es unumgänglich, aus den inzwischen mehrere Gigabyte umfassenden Log – Dateien Informationen performant und schnell auszulesen. Hierfür werden bei der dsb AG Skripte geschrieben, welche nur die relevanten und für den Administrator interessanten Informationen aus den Log – Dateien ziehen und ausgeben. Durch die oftmals wechselnde Struktur der Dateien und der geänderten Anforderungen der Administratoren müssen diese Skripte schnell und einfach zu bearbeiten sein. Darum werden in der Abteilung diverse Skriptsprachen verwendet, unter anderem Python und Perl.

2 Ausführlichere Informationen zur dsb AG unter www.dsb.net

2. Skriptsprachen im Allgemeinen

2.1 Definition von Skriptsprachen

Skriptsprachen sind Programmiersprachen, welche vor der Ausführung des Programm - codes nicht kompiliert werden müssen. Sie werden zur Laufzeit von einem Interpreter interpretiert. Ursprünglich waren Skriptsprachen im Sprachumfang deutlich schlanker als Programmiersprachen wie Pascal oder C.

Aber alleine die Tatsache, dass eine Sprache interpretiert wird, ist kein ausreichendes Kriterium für eine Skriptsprache. So gibt es auch klassische Programmiersprachen, welche einen Interpreter verwenden.

Versucht man eine Sprache anhand ihres Sprachumfangs zu klassifizieren, stößt man auch hiermit schnell auf Probleme. Skriptsprachen haben im Laufe der Zeit ihre Funktionalität erweitert und stehen den klassischen Programmiersprachen in nichts nach. So zählt man heutzutage einige Skriptsprachen schon zu vollwertigen Programmier – sprachen, zum Beispiel Perl.

Die Abgrenzung zwischen klassischer Programmiersprache und Skriptsprache scheint zu verschwimmen. In immer mehr Fällen ist eine eindeutige Klassifizierung mithilfe der formellen Fakten nahezu unmöglich. In solchen Fällen erreicht man eine klare Abgrenzung am ehesten durch die unterschiedlichen Einsatzgebiete.

2.2 Geschichtliche Entwicklung

Die allerersten „Skripte“ gab es schon bei den Lochkarten. Damals wurde der zu verarbeitende Code in Karten gestanzt, welche dann von einer Lochkartenanlage gelesen wurden. Dabei enthielt der oberste Teil eines zusammengehörigen Stapels nötige Steuerbefehle. Diese Skripte wurden dann von JCL (Job Control Language) abgelöst, welche von IBM für ihren IBM 360 in den sechziger Jahren erfunden wurde.

Für UNIX gab es Kommandosprachen wie sh und awk. Sie sind die Schnittstellen zwischen Benutzer und Betriebssystem. Diese Sprachen erhielten im Laufe der Zeit unter anderem Variablen und Schleifenanweisungen. Dadurch konnte der Anwendungsbereich der Sprachen ausgebreitet werden.

So wurde Perl mit dem Ziel entwickelt, awk als Kommunikationsmittel auf UNIX – Systeme abzulösen, inzwischen ist es aber eine der führenden Sprachen bei der Programmierung größerer und kleinerer Skripte.

2.3 Merkmale einer Skriptsprache

Zwischen den einzelnen Skriptsprachen sind natürlich Unterschiede vorhanden, so kann man nicht alle in einen Topf werfen. Doch bestimmte Merkmale sind bei sehr vielen Sprachen vorhanden und können so als Merkmale aufgezählt werden.

- Interpretiert
  Der Quellcode eines Programmes wird von einem Interpreter zur Laufzeit ausgewertet und interpretiert. Der Zwischenschritt über einen Compiler entfällt.
- Fehlertoleranz
  Erst wenn der konkrete Teil eines Skriptes ausgeführt wird, werden die Fehler dort erkannt.
- Typenkonversion
  Bei Skriptsprachen ist diese meist dynamisch und automatisch. Somit müssen Variablen nicht deklariert werden und können zur Laufzeit ihren Datentyp ändern.
- Dynamik
  Quelltext kann während der Laufzeit übersetzt werden. Dadurch kann ein Programm zur Laufzeit Quelltext erzeugen, einlesen und diesen dann direkt ausführen.

3. Grundlagen von Python [3]

„Die Grundlage ist das Fundament der Basis.“ [4]

3.1 Was ist Python?

Python ist eine interpretierte und objektorientierte Skriptsprache. Wie kaum eine andere Sprache verbindet Python einen großen Sprachumfang mit einer klaren und logischen Syntax, welche gerade für Programmieranfänger leicht zu erlernen ist. Dadurch erreicht man schon nach kurzer Zeit erste Lernerfolge durch eigene Skripte.

Aber auch fortgeschrittene Programmierer werden sich mit Python anfreunden können, beinhaltet es so wichtige Features wie Klassen, Module und Ausnahmen, um nur einige zu nennen.

Pythons Entwicklung startete 1990 im „Centrum voor Wiskunde en Informatica“ in Amsterdam und wurde von Guido van Rossum iniziiert. Seither ist Python im Besitz der „Python Software Foundation“, welche die Weiterentwicklung überwacht und vorantreibt.

Inzwischen ist Python für die geläufigsten Plattformen verfügbar, unter anderem für viele Unix – Derivate, Windows, OS/2, Mac und Amiga. Da der Sourcecode von Python frei verfügbar ist, wird man im Internet auch Versionen für andere, exotischere Plattformen finden.

3.2 Aufrufen und Benutzen des Python – Interpreter

Den Python – Interpreter kann man, bei korrekter Installation von Python, in der Shell mit dem Kommando „python“ aufrufen.

Man kann den Interpreter auf drei verschiedene Arten starten:

3.2.1 Interaktiver Modus

In den interaktiven Modus von Python gelangt man, indem der Interpreter nur mit dem Kommando „python“ gestartet wird. Zuerst werden diverse Informationen über die Python – Version und das Betriebssystem auf dem Bildschirm ausgegeben, danach wird der Prompt angezeigt (normalerweise drei „<“). Jetzt liest der Interpreter die eingegebenen Kommandos und verarbeitet sie, ähnlich einer Unix – Shell.

3 Ausführliche Einführung, geschrieben von Guido van Rossum, unter: http://docs.python.org/tut/tut.html
4 Le Corbusier, französisch – schweizerischer Architekt (1887 – 1965)

3.2.2 Kommandoaufruf

Wird der Interpreter mit „python -c *command*“ aufgerufen, so startet der Interpreter, arbeitet das Kommando „*command*“ ab und wird danach wieder beendet. Da An - weisungen in Python oftmals Leerzeichen enthalten, sollte man das Kommando in Anführungszeichen schreiben.

3.2.3 Skriptaufruf

Ein Python – Skript wird mit dem Befehl „python *file*“ gestartet. Hier wird nun der Interpreter gestartet, welcher dann das Skript verarbeitet, um danach wieder beendet zu werden.

3.3 Syntaktische Besonderheiten

Die auffälligste Besonderheit wird jedem, der schon einmal programmiert hat, sofort auffallen. Blöcke werden bei Python nicht mit Start – und Endmarkierungen wie Klammern ({...}) gekennzeichnet, sondern sind alleine durch das Einrücken mit Tabulatoren definiert. Alles, was auf einer Einrückungsebene steht, gehört somit zu einem Block. Alleine diese Tatsache schreckt schon viele Umsteiger ab, die damit nicht zurechtkommen oder nicht zurechtkommen wollen.
Doch gerade diese Eigenschaft ist einer der Garanten für einen gut strukturierten und leserlichen Code. So gewöhnt man sich als Anfänger gleich einen sauberen und schönen Programmierstil an. Aber auch Umsteiger, welche bisher keinen besonderen Wert auf Einrückungen gelegt haben, werden nach kurzer Zeit diese Art der Blockbehandlung nicht mehr missen wollen.

Eine weitere Eigenheit von Python ist das fehlende Semikolon am Ende einer Anweisung. Python interpretiert das Zeilenende als Ende einer Anweisung. Somit steht immer nur eine Anweisung in einer Zeile. Das widerrum fördert die Lesbarkeit des Programmes. Zwar kann man im Notfall mehrere Anweisungen in eine Zeile schreiben, indem man sie mit einem Semikolon trennt, doch wird dies ausdrücklich nicht empfohlen.

3.4 Hello World

Jede mögliche Einführung in Sprachen beginnt meistens mit dem klassischen „Hello World“ - Programm. In Python ist dieses Skript besonders schnell und einfach geschrieben. Dazu reicht einfach diese Anweisung:

```
print 'Hello World'
```

Listing 3.4.1 – Hello World

3.5 Datentypen

Python kennt keine Variablendeklaration. Wenn man eine Variable braucht, erstellt man sie einfach. Dabei muss man sich auch nicht überlegen, ob eine ganzzahliger Datentyp reicht, oder ob man später noch Fließkommazahlen dafür verwenden muss. Das erledigt alles Python. Python weist den Variablen dynamisch den passenden Datentypen zu, welcher sich auch zur Laufzeit ändern kann. So ist folgendes Coding zulässig:

```
var = 9                         # integer ( 9 )
var = var / 2                   # float ( 4.5 )
var = str(var) + 'Meter'        # String ( 4.5 Meter )
```

Listing 3.5.1 – Beispiel für dynamische Datentypen

Dies ist ein großer Vorteil gegenüber vielen anderen Programmiersprachen wie Java und Pascal. Bei diesen Sprachen müssen die Variablen schon vor der Benutzung deklariert sein. Auch lässt sich der Datentyp nicht ohne weiteres verändern.

Python unterstützt die typischen Datentypen wie Ganzzahlen (integer), Fließkommazahlen (float) und Zeichenketten (string). Zu Erwähnen ist hier der Datentyp „long integer“, der Ganzzahlen mit beliebig vielen Stellen enthalten kann. Diese sind nur durch den Speicher begrenzt. Ein weiterer Datentyp sind die komplexen Zahlen (complex), welche nur selten bei Skriptsprachen zu finden sind.

3.5.1 Sequenzen (Listen und Tupel)

Dieser Datentyp wird unterteil in zwei Untertypen: Listen (list) und Tupel (tuple). Listen bieten eine große Anzahl an Funktionen an, wie Sortieren, Löschen, Wiederholen, Selektieren, Konkatenieren und weitere. Dabei können Listen beliebige Objekte enthalten, somit auch weitere Listen.

Tupel sind vom Aufbau her identisch mit Listen. Im Gegensatz zu Listen sind Tupel dagegen unveränderbar. Einmal erstellt lassen sie sich ohne Neuanlegen der Tupel nicht modifizieren.

Indexe innerhalb Sequenzen starten bei null. Dies ist besonders beim Selektieren einzelner Objekte oder Bereiche innerhalb einer Sequenz wichtig.

Im folgenden Listing wird die Funktionalität von Sequenzen dargestellt, jeweils mit erklärenden Kommentaren dazu.

```
Liste = []                                  #Liste wird erstellt
Liste.append(5)                             #Die Zahl '5' wird angehängt
Liste += [6]                                #Die Zahl '6' wird angehängt
print Liste*3                               # gibt '[5,6,5,6,5,6]' aus (Ausgabe der Liste x 3)

Liste = range(6)                            #Liste wird erstellt mit 6 Elementen [0,1,2,3,4,5]
                                            # range(x) erstellt eine Liste von 0 bis x-1
Liste2 = [11,12]                            #Zweite Liste wird erstellt

print Liste[2:5]                            #gibt '[2,3,4]' aus, 'von:bis' exklusive 'bis'

Liste[1:3] = Liste2
print Liste                                 #gibt '[0,11,12,3,4,5]' aus
print Liste[1]                              #gibt '11' aus

Tupel = (1,2,3)
Tupel[1] = 5                                #geht nicht!
Tupel += (4)                                #geht! Tupel wird neu angelegt

Liste = range(10)
print Liste[::2]                            #gibt [0, 2, 4, 6, 8] aus (2-er Schritte)
```

Listing 3.5.1.1 – Möglichkeiten von Sequenzen

Neben append() gibt es noch weitere Funktionen, um Listen zu manipulieren. So zum Beispiel extend(), insert(), remove(), sort() und count()[5].

3.5.2 Dictionaries

Ein weiterer Datentyp von Python ist das sogenannte Dictionary. Im Gegensatz zu Sequenzen werden hier die Elemente nicht mit Zahlen indiziert, sondern mit Schlüssel, welche eine unveränderliche Form besitzen. Also können sie zum Beispiel mit Strings, Zahlen oder auch Tupel indiziert werden. Listen dagegen können nicht verwendet werden, da diese ja manipuliert werden können.

Somit bestehen Dictionaries aus *Schlüssel – Wert* Paaren. Mithilfe der Schlüssel können dann jeweils die dazugehörigen Werte ausgelesen werden. Versucht man, einen Schlüssel zwei mal zu vergeben, so wird der alte Schlüssel verworfen.

```
Alter = { 'Tom' : 32, 'Petra' : 28, 'Josef' : 44}   #Dictionary wird erstellt
Alter['Petra'] = 29                                 #Petra wird älter
print Alter                                         #gibt ['Tom' : 32, 'Petra' : 29, 'Josef' : 44] aus

del Alter['Josef']                                  #Josef ist gestorben

print Alter.keys()                                  #gibt ['Tom', 'Petra'] aus
```

Listing 3.5.2.1 – Arbeiten mit Dictionaries

5 „Python 2.4 Quick Referenze“, Richard Gruet

3.6 Kontrollstrukturen

3.6.1 Bedingungen

Wie in anderen vielen gängigen Sprachen auch, kennt Python Bedingungen in Form von *if ... elif ... else.*

```
a = 5
b = 6
c = 7
if a>b:
        print 'a ist groesser wie b'
elif a<b:
        print 'a ist kleiner wie b'
else:
        print 'a und b sind gleich groß'

if a<b<c:
        print 'b liegt zwischen a und c'
```

Listing 3.6.1.1 – Beispiel für Bedingungen

Bei diesem Listing fallen die unnütz wirkenden Doppelpunkte am Ende jeder Bedingung auf, welche gerade für Neulinge eine potentielle Fehlerquelle darstellen. Doch sind die Doppelpunkte nicht nur zur Verwirrung da. So haben Programmierer die Möglichkeit, direkt hinter der Bedingung eine einzelne Anweisung zu schreiben, um somit einen etwas kompakteren Quellcode zu erhalten. Darunter leidet jedoch die Lesbarkeit.
Ein negiertes *if* ist auch vorhanden, in Form von *if not.*

3.6.2 Schleifen (*while* und *for*)

Python unterscheidet zwischen 2 Schleifenarten: die *while* – Schleife und die *for* -Schleife. Die *while* – Schleife funktioniert in Python wie in vielen anderen Sprachen auch. Sie hat Eintrittsbedingungen, zusätzlich zu anderen Sprachen aber auch noch die Möglichkeit, ein optionales *else* nach der *while* – Schleife anzubauen, vergleichbar mit dem *else* bei der *if*– Bedingung. Dieser Zweig wird einmalig durchlaufen, sollte die Eintrittsbedingung nicht stimmen.
In Python können *for* – Schleifen über jede beliebige Folge laufen, also nicht nur über einen bestimmten Zahlenbereich, wie es bei anderen Sprachen üblich ist. Die Schleife besteht aus einer Laufvariablen und einer Folge in form einer Liste oder eines Tupels.

```
a = 1
while a <= 10:
        print a
        a += 1
else
        print 'a ist größer als 10'

#gibt '1 2 3 4 5 6 7 8 9 10 a ist größer als 10' aus
```

```
for z in range(10):
        print z

# gibt 0 1 2 3 4 5 6 7 8 9 aus
```

Listing 3.6.2.1 – while- und for- Schleifen

Im folgendem Listing werden verschachtelte *for* – Schleifen und *if* – Bedingungen benützt, um Primzahlen von 2 bis 1000 anzeigen zu lassen.

```
p = range(2,1001)
for u in xrange(2,33):
        if u in p:
                for s in xrange(u*2,1001,u):
                        if s in p:
                                p.remove(s)
print p
```

Listing 3.6.2.2 – Beispiel für Schleifen und Bedingungen

3.7 Zugriff auf Datenbanken

Um mit Python auf Datenbanken zugreifen zu können, benötigt man zusätzliche Module. Hat man aber das notwendige Modul eingebunden, gestaltet sich die Benutzung der Datenbank sehr einfach.

3.7.1 Beispiel MS Access

Möchte man mit Python eine MS Access – Datenbank ansprechen, so benötigt man ein entsprechendes Modul. Ein solches Modul wird zum Beispiel von den „Python for Windows Extensions“[6] bereitgestellt. Diese benutzen den ODBC – Datenbanktreiber von Windows. Nachdem man die Extension installiert hat, muss man die Access – Datenbank (Dateiendung .mdb) beim ODBC-Datenbanken-Administrator eintragen. Dieser ist unter Systemsteuerung -> Verwaltung zu finden.

Beispiel:

```
Import dbi, odbc                                        # laden der nötigen Module
db = odbc.odbc('DatenBankName')                         # Verbindung zur Datenbank

cursor = db.cursor()                                    # 'öffnen' der Datenbank
cursor.execute('SELECT * FROM %s' % (TabellenName))     # SQL-Statement ausführen
data = cursor.fetchall()                                # Ergebnisse holen
for index, datum  in enumerate(data):                   # Ergebnisse ausgeben
        print index, datum

cursor.close()                                          # Datenbank 'schließen'
db.close()                                              # Verbindung zur Datenbank beenden
```

Listing 3.7.1 Beispiel für Datenbank-Zugriff

6 „Python for Windows Extensions“, http://starship.python.net/crew/mhammond/win32/

4. Skriptprogrammierung bei der dsb AG

4.1 Allgemeine Nutzung von Skripten

Die Abteilung dezentrale Systeme der dsb AG verwendet Skripte hauptsächlich zur Automatisierung von Routineaufgaben. Ein wichtiger Vorteil von Skriptsprachen ist das Verändern der Skripte, ohne einen Compiler. Somit kann man schnell seinen Code manipulieren und testen und kann so produktiver arbeiten. Die Zwischenschritte über einen Compiler bremst dagegen den Programmierer ab, der bei der kleinsten Änderung sein ganzes Skript neu compilieren muss. Dazu werden hauptsächlich die Sprachen Perl und Python benutzt.

Jedoch ist bei diesen Sprachen immer ein Interpreter notwendig, der leider nicht auf allen Maschinen zur Verfügung steht und auch nie zur Verfügung stehen wird. So weicht man hier auf Shell – Skripte aus.

4.2 Problemstellung

Die dsb AG hält für ihre Kunden ISDN – Verbindungen bereit, über die sie sich auf die Server in der Firma einwählen können. Da diese Verbindungen aber keine Flatrates sind, werden sie natürlich akribisch genau aufgezeichnet. In den entstehenden Log – Dateien kann man genau nachvollziehen, wann ein User wie lange eine Verbindung benutzt hat. Nun können diese Log – Dateien schnell 100 Megabyte überscheiten und haben auch des öfteren über eine Million Zeilen. Diese jetzt manuell auszulesen ist umständlich und zeitaufwendig.

In Listing 4.2.1 sehen sie einen Ausschnitt aus solch einer Log – Datei. Einzig die fettgedruckte Zeile interessiert hier den Administrator. Zeilen mit diesem Muster müssen ausgelesen, verarbeitet und formatiert dargestellt werden.

```
Jan  1 13:04:42 10.4.49.10 last message repeated 2 times
Jan  1 13:04:44 arcorvpn 13:04:44 ACCT: INET: 01.01.2005 13:04:21 0 50 194.12.208.5:0/204 ->...
Jan  1 13:04:44 10.254.111.1 13:04:44 INET: NAT: refused incoming session on ifc 100 prot 17...
Jan  1 13:04:45 arcorvpn 13:04:45 PPP: ppppille: incoming connection closed, duration 3092
sec, 151124 bytes received, 3160033 bytes sent, 0 charging units, 0 charging amounts
Jan  1 13:04:46 10.254.111.1 13:04:46 INET: NAT: refused incoming session on ifc 100 prot 17...
```

Listing 4.2.1 – Ausschnitt aus einer Log – Datei

Dabei ist es wichtig, dass die Verbindungsdauer einzelner Personen addiert werden und eine sinnvolle Formatierung der Zeitangabe durchgeführt wird.

Da die einzelnen Log – Dateien nichtssagende Namen haben, werden auch die Informationen über den Aufzeichnungsstart und das Aufzeichnungsende benötigt.

4.3 Lösungsansatz

Zuallererst musste ein Konzept ausgearbeitet werden, wie das fertige Ergebnis aussehen sollte. Also beginnend vom Aufruf des Skriptes bis hin zur Ausgabe der Auswertungen einer Log – Datei.

Da in der Abteilung sehr viel mit der Konsole gearbeitet wird, entschied man sich für ein Skript, welches von der Konsole aus aufgerufen wird und dort auch seine Ergebnisse präsentiert. Diese Ergebnisse sollten den Namen des Users ausgeben, der eine ISDN – Verbindung benutzt hat, sowie die insgesamte Dauer aller Verbindungen eines Users, welche in der Log – Datei erfasst wurden. Zusätzlich soll noch der erfasste Zeitraum ausgegeben werden.

Im ersten Schritt wird die Datei zeilenweise eingelesen. Jede Zeile wird vom Skript untersucht und analysiert. Das Skript geht dazu Zeile für Zeile durch und schaut, ob sie relevant ist oder nicht. Die relevanten Zeilen werden temporär in einem Array gespeichert. Gleichzeitig werden die Zeilen noch mehrmals geteilt, um somit auf die einzelnen Daten wie Name und Dauer der Verbindung Zugriff zu haben.

Sollte der Name, welcher in der Zeile steht, schon vorhanden sein, so wird nicht ein neues Element in das Array eingefügt, sondern die Dauer der Verbindung zum schon bestehenden Datensatz addiert.

Nachdem die Datei eingelesen wurde und alle relevante Daten in einem Array stehen, erfolgt nun die Ausgabe der Daten. Diese muss alle wichtige Informationen liefern, dabei aber auch übersichtlich und klar strukturiert sein. Dabei muss vorallem auf die Ausgabe der Verbindungsdauer geachtet werden. Allein die Ausgabe der Sekunden ist nicht benutzerfreundlich. Darum werden die Zeitangaben noch formatiert, sodass der Benutzer eine komfortablere Ausgabe erhält.

4.4 Ergebnis

Dieses Ergebnis erhält nun der Benutzer des Skriptes:

```
                Beginn [01.Aug Uhrzeit 00:57:56]
                Ende   [31.Aug Uhrzeit 17:24:39]

    antwerpen:                  18min 12sec
    frankfurt:                  8min 11sec
    grosslehna:                 2h 30min 30sec
    huerth:                     6sec
    koelnniehl:                 39sec
    ludwigshafen:               23sec
    marl:                       1min 5sec
    ppp-th-ki-gaggenau:         58sec
```

5. Literaturverzeichnis

5.1 Bücher

- **Mark Lutz:** „Python kurz und gut“. (3. Auflage Mai 2005)

5.2 Internet

- **Mark Pilgrim:** „Dive into Python“. (Version 5.4, Mai 2004)
  Link: http://www.diveintopython.org/toc/index.html
- **Richard Gruet:** „Python 2.4 Quick Reference“.
  Link: http://rgruet.free.fr/PQR24/PQR2.4.html

6. Anhang

6.1 Python – Skript

```
# Globale Variablen und Einstellungen

pfad = "/tmp/logfiles/mai04/"       # relativer Pfad zu den Log-Files

temp = ['tada'];
datenbank = {}
import re, os, string
p= re.compile('incoming connection closed')

filelist = os.listdir(pfad)
for filename in filelist:
        if re.compile('messages').search(filename):
                f1=open(pfad+filename,'r')
                ersteZeile1 = f1.readline()
                f1.close
                if(ersteZeile1.find("  ",3,5)!=-1):
                        ersteZeile1 = ersteZeile1.replace("  "," 0",1)
                ersteZeile = ersteZeile1.split(" ")
                print "\n\t\t\tBeginn ["+ersteZeile[1]+"."+ersteZeile[0]+" Uhrzeit "+ersteZeile[2]+"]";

                f=open(pfad+filename,'r')
                while 1:
                        zeile = f.readline()
                        if(zeile.find("  ",3,5)!=-1):
                zeile = zeile.replace("  "," 0",1)
                        if not zeile:
                                break
                        if p.search(zeile):
                                temp[0] =  zeile.split(" ")
                                if datenbank.has_key(temp[0][6])==1:
                                        datenbank[temp[0][6]] = int( temp[0][11])+int(datenbank[temp[0][6]])
                                else:
                                        datenbank[temp[0][6]] = temp[0][11]
                        letzte = zeile
                f.close()
                letzteZeile = letzte.split(" ")
                print "\t\t\tEnde   ["+letzteZeile[1]+"."+letzteZeile[0]+" Uhrzeit "+letzteZeile[2]+"]\n"
                items=datenbank.items()
                items.sort()
                for name,value in items:
                        value = int(value)
                        if(value>3599):
                                dauer_stu = (value-(value % 3600))/3600
                                tempo = value - 3600*dauer_stu
                                dauer_min = (tempo-(tempo % 60))/60
                                dauer_sec = tempo % 60
                                dauer = str(dauer_stu)+"h "+str(dauer_min)+"min "+str(dauer_sec)+"sec"
                        elif(value<3600 and value>59):
                                dauer_min = (value-(value % 60))/60
                                dauer_sec = value % 60
                                dauer = str(dauer_min)+"min "+str(dauer_sec)+"sec"
                        else:
                                dauer = str(value)+"sec"
                        if(len(name)<8):
                                tabs =  "\t\t\t\t\t"
                        elif(len(name)>15):
                                tabs =  "\t\t\t"
                        else:
                                tabs =  "\t\t\t\t"

                        print "\t%s%s%s" % (name, tabs, dauer)
                datenbank.clear()
```