

# e-Mail mit Python

## Wer hat die e-Mail erfunden?

Das war der Informatiker Ray Tomlinson. Er ist auch verantwortlich für die Verwendung des at-Zeichens @ als wesentlichem Teil einer Adresse.

Lange bevor das Internet im heutigen Sinn existierte, gab es das ARPANET (das waren amerikaweite Telefonleitungen des Militärs, die allerdings nicht wirklich verwendet wurden und von der Regierung einem sinnvolleren Nutzen zugeführt werden sollten). Es wurde 1971 vom amerikanischen Verteidigungsministerium eingerichtet und bestand vorerst zu Testzwecken aus 15 durch ein rudimentäres Netzwerk verbundenen Universitäten.

Tomlinson war es gewohnt, kurze Nachrichten und Memos innerhalb seiner Universität zu verschicken und fragte sich, warum er diese nicht auch an Leute an anderen Universitäten verschicken könnte. Wie sollte man aber wissen, ob eine Notiz für Adressaten innerhalb oder außerhalb der lokalen Uni gedacht war? Er wählte einfach das Zeichen @ (dieses gab es schon lange!) um Nachrichten nach draußen kenntlich zu machen.

Und was war die erste e-Mail?

Leider nichts so großartiges wie 'What has God wrought?' von Samuel Morse oder 'Watson, come here, I want you' von Graham Bell.

Tatsächlich war sie an ihn selbst gerichtet, allerdings am zweiten Computer in seinem Zimmer. Die Test-Mail hatte, wie er sich erinnert, den Inhalt 'QWERTYUIOP'.

Vielleicht wird es Dich verwundern – wenn Du im Verlauf unserer Nachforschungen den inneren Aufbau heutiger e-Mails kennlernst: da ist alles wie damals! Kein Attachment, keine Sonderzeichen, keine Umlaute,... alles wird in Form von simplen 7-bit ASCII-Zeichen kodiert!

## Python als Postbote

Wir werden e-Mails lesen und versenden, OHNE ein spezielles Programm dafür in Anspruch zu nehmen. Also kein Outlook, kein Browser, kein Mail-Client. Python stellt uns alle nötigen Werkzeuge zur Verfügung.

Freilich haben große Mail-Programme auch ihre Vorteile: Adressbücher, der einfache Umgang mit Attachments, das Speichern von erhaltenen Mails, HTML-Mails – das könnten wir in Python aber natürlich auch alles nachbilden. Uns geht es jetzt aber um die Grundlagen.

Und diese Grundlagen sind interessant! Was ein großes Programm nämlich verschweigt, ist die **tatsächliche Kommunikation** unseres Computers mit dem Mailserver draußen in der Welt. Mit Python bekommen wir hier einen Einblick und können diese Kommunikation Schritt für Schritt mitverfolgen. Mail-Programme verstecken die Tatsache, dass hier zwei Computer miteinander 'sprechen' völlig und reduzieren alles auf das Klicken eines Knopfes. Doch was dahinter passiert, ist alles andere als langweilig! Und mit Python können wir das Schritt für Schritt interaktiv mitverfolgen.

Die Kommunikationsvorschriften, um die es hier geht sind:

**POP** (Post Office Protocol, Version 3) zum Abholen der Mails

**SMTP** (Simple Mail Transfer Protocol) zum Verschicken von Mails

Gleich vorweg eine kleine Warnung bzw. Einschränkung: Da mit e-Mails viel Unfug getrieben werden kann (Spam-Mails,...), haben viele Provider Vorkehrungen gegen Missbrauch getroffen. Selbstverständlich würdest *Du* es aus moralischen Gründen *nicht* tun, doch gibt es finstere Existenzen, die es systematisch versuchen, und mit Python wäre es prinzipiell auch ganz leicht möglich, folgendes zu tun:

- Hundert Mails pro Sekunde zu verschicken: SMTP ist in der Standardversion eine sehr unsichere Sache, da der Server keine Anmeldung des Benutzers erfordert. Provider erzwingen daher oft eine Technik namens 'SMTP after POP'. Erst wenn man sich beim Postabholen identifiziert hat, ist der Post-Versand für einige Minuten offen. Oder ein moderner Provider verlangt auch für SMTP eine Zugangskontrolle (diese erweiterte Funktionalität ist ebenfalls in Python enthalten).
- Hundert mal pro Sekunde nachzufragen, ob neue Post da ist: Deshalb erzwingen Provider oft eine Wartepause zwischen Verbindungsaufbauten, damit ihre Server nicht überlastet werden.
- Die Verbindung 24 Stunden täglich offen zu lassen und damit für andere Nutzer zu blockieren: nach einer Maximalzeit wird die Verbindung automatisch beendet.

Also gibt es oft kleine Unterschiede in der Art, wie man mit seinen Mails umzugehen hat. Das große Mail-Programm erledigt das alles automatisch (etwa: Mail schicken? Zugang geschlossen? Kein SMTP-Passwort? Also schnell eine POP-Abfrage im Hintergrund einschieben).

Aber für echte Pythonisten ist das alles kein Problem. Auch über spezielle Arten der Kommunikation (gesicherte Anmeldung/Übertragungen usw.) gibt es Informationen wie gewohnt in der Beschreibung der entsprechenden POP und SMTP Module.

Welche Informationen brauchst Du: Die Internetadressen der Mailserver, deinen Benutzernamen und das Passwort. Beachte, dass der Benutzername meist anders lautet, als deine e-Mailadresse!

Meine Tabelle sieht etwa so aus:

Mailadresse:	<a href="mailto:Wolfgang.Urban@work.at">Wolfgang.Urban@work.at</a>
POP Server:	pop3.schule.at
POP Username:	wupopper
POP Passwort:	secret
SMTP Server:	mailer.schule.at
SMTP Username:	wumailer
SMTP Passwort:	topsecret

## Python holt die Post

Die dafür nötigen Dinge sind im Modul poplib versammelt.

```
>>> import poplib
```

Nun möchte ich eine Verbindung zu meinem POP-Server herstellen. Python verwendet dazu ein Objekt der Klasse POP3

```
>>> server = poplib.POP3('pop3.schule.at')
```

Mal sehen, wie der Server uns begrüßt:

```
>>> server.getwelcome()
+OK Welcome at schule.at POP3 Server V0.7.0 <20894.178565@mp07rz5>
```

Fein! Wir sind drin, der Server hat uns begrüßt  
Jetzt werden wir uns identifizieren:

```
>>> server.user('wupopper')
+OK May I have your password, please?
```

Na gut, hier kommt es (der Unterstrich ist nötig, da 'pass' ein Python-Befehl wäre):

```
>>> server.pass_('secret')
+OK mailbox has 6 messages (1739 octets)
```

Wunderbar! Und beachte bitte – die Textzeilen mit dem + davor sind keine Python-Meldungen. Du hast dich wirklich direkt mit dem Mailserver unterhalten. Und er hat auch gleich mitgeteilt, wie viel Post da ist. In meinem Fall 6 e-Mails.

Hättest Du '0 Messages' gelesen, ist dein Konto momentan leer. Dann solltest Du dich sofort mittels

```
>>> server.quit()
```

wieder abmelden, um Kapazität für die übrigen Kunden frei zu machen. Versäumst Du dies, schließt der Mailserver automatisch nach einiger Zeit die Verbindung (und brummt dir eventuell ein paar Strafinuten Wartezeit auf, bis er dich wieder akzeptiert. Dann würdest Du bei getwelcome() so etwas lesen wie 'connection delayed for 5 minutes' oder ähnlich.

Angenommen, Du hast Post und willst sie lesen. Meine 6 Mails wären jetzt mit Nummern 1 bis 6 versehen und bereit zur Abholung.

```
>>> server.list()
('+OK scan list follows', ['1 875', '2 541776', ..... '6 921'], 1739)
```

Python hat ein Tupel geliefert: Einen Meldungs-String, eine Liste, und eine (für uns uninteressante) Zahl. Ganz wichtig ist die Liste: hier ist vermerkt, wie groß die einzelnen Mails sind. Die erste Mail ist wohl ein kurzer Text, die sechste auch. Mail 2 ist ein halbes Megabyte groß – da hat vermutlich jemand ein Bild, ein Programm oder eine andere Datei als Attachment mitgeschickt?

Von großen Mails lassen wir jetzt erstmal die Finger. (Was MIME-Typen sind und was base64-Encoding bedeutet, erfährst Du im Unterricht. Aber auch dort wirst Du die Empfehlung hören: das geht für Nicht-Pythonprofis mit dem großen Mailprogramm einfacher. Allerdings verbietet Dir niemand, die Dokumentation der Python Mime-Bibliotheken zu lesen – schwierig ist es nicht!)

Ich will jetzt aber die letzte Mail lesen, sie hat Nummer 6. retr() steht für 'retrieve' (diese Kürzel stammen alle noch aus den Urtagen des Internet! Und weil sie jeder Programmierer inzwischen kennt, ändert man sie auch nicht):

```
>>> header, message, octets = server.retr(6)
>>> print message
```

Was jetzt kommt, erspare ich Dir zu lesen. Ein Haufen technischer Details (Absender, Subject,

Timestamp, Typ..., und danach der Text der e-Mail. Du probierst es ja hoffentlich ohnehin selbst aus. Wovon das wichtigste ist: Datei aufmachen, message reinschreiben, Datei schließen, und diese 3 Zeilen Python-Code speichern Deine Mail dauerhaft ab.

Diese Mail ist nun gelesen, ich brauch sie nicht mehr, und kann sie löschen:

```
>>> server.delete(6)
```

Jetzt ist sie markiert (widerrufbar!) als wegzuwerfen. Erst wenn ich die Verbindung später schließe, wird die Mail tatsächlich gelöscht (unwiderruflich).

Jetzt reicht's mir, ich schließe die Verbindung:

```
>>> server.quit()
```

## Nachsatz:

Wenn ich wissen wollte, was da in der Mail Nummer 2 riesiges verborgen ist, kann Python das natürlich auch, ohne dass man vorher alles einlesen muss. Du findest in der poplib-Modul-Beschreibung die Möglichkeit, nur den 'Header' der Mail, oder nur einige Zeilen (etwa die ersten 50) abzuholen. Da findest Du sicher den Betreff und die Beschreibung des Attachments. Das ist deswegen einfach möglich, da Attachments beim Verschicken in eine Art 'Pseudo-Textzeichen-String' verwandelt werden, die dann als normaler e-Mail-Text angehängt sind. Probier es aus – schick Dir selbst ein kleines Bild (nur wenige (Kilo)byte!) als Attachment und schau Dir (wie oben vorgeführt) einfach die ganze Message in Python an. Du lernst dabei, wie man es schafft, dem POP-Protokoll, das eigentlich nur mit einem einzigen Textstring arbeiten kann, beliebig viele Dateien beliebigen Inhalts unterzujubeln....

## Python verschickt Post

Nun wollen wir eine e-Mail erzeugen und verschicken. Dabei ist zu berücksichtigen, dass wir auf einem ganz elementaren Niveau mit dem Server kommunizieren, wir müssen uns genau an die Regeln halten, nach denen eine e-Mail zu formatieren ist.

Vorerst importieren wir das passende Python-Modul und die Zeit-Bibliothek:

```
>>> import smtplib
>>> import time
```

Einige Einstellungen werde ich gleich in Variable verpacken, um später weniger tippen zu müssen:

```
>>> absender = 'Wolfgang.Urban@work.at'
>>> adressat = 'cynthia.python@python.at'
>>> betreff = 'Einladung'
>>> inhalt = 'Hallo Cynthia!\nVergiss nicht auf unsere Party!'
>>> zeit = time.ctime(time.time())
```

Jetzt baue ich die Mail nach Vorschrift zusammen. Alles kommt in **eine einzige** Stringvariable hinein!

```
>>> text = 'From: '+absender+'\n'  
>>> text += 'To: '+adressat+'\n'  
>>> text += 'Date: '+zeit+'\n'  
>>> text += 'Subject: '+betreff+'\n'
```

Das waren die Formalitäten (weitere Informationen wie 'ReplyTo:' wären natürlich auch möglich gewesen – genau so, wie wir sie vorhin beim POP3-Empfang gelesen haben), nun hänge ich den Inhalt meiner Mail dran.

```
>>> text = text + inhalt
```

Zur Kontrolle:

```
>>> print text
```

Das wars. **Und jetzt ab die Post!** (Ich schreib hier keine Rückmeldungen des Servers mehr hin – Du probierst das ja ohnehin selbst aus)

Eine Verbindung zum Server aufbauen:

```
>>> server = smtplib.SMTP('mailer.schule.at')
```

Mein Server verlangt eine Authentifizierung, bei Dir kann das anders sein:

```
>>> server.login('wumailer', 'topsecret')
```

Und jetzt übertragen:

```
>>> server.sendmail(absender, adressat, text)
```

Fertig!

```
>>> server.quit()
```

## Nachsatz:

Wenn Du mit Python öfter mailen willst, dann schreibst Du Dir am besten ein kleines Programm, in dem die fix bleibenden Werte gespeichert sind (besser natürlich ohne Passwort – das holst Du Dir mittels `raw_input`)

```
class Mail():  
    def __init__():  
        self.adresse = 'Wolfgang.Urban@work.at'  
        self.popserver = 'pop3.schule.at'  
        self.popuser = 'wupopper'  
        ....und so weiter
```

verwendest dann

```
>>> ich = Mail()
```

und kannst dann einfach etwa

```
>>>server.user(ich.popuser)
```

verwenden, ohne alle Details im Kopf haben zu müssen.

Aber noch viel besser ist natürlich der Ausbau der Klasse um einige Methoden, wie

- Verbindung aufbauen
- Mailanzahl und Größen holen
- Mail Nummer n anzeigen
- Verbindung beenden
- Status abfragen (ob die Verbindung noch steht oder nicht)

und um eine Klasse (mit einem Python-Dictionary) als Adressbuch

(Deine Benutzernamen und Passworte könntest Du sinnvollerweise in einer Textdatei auf der Festplatte (oder einer privaten Diskette, vielleicht sogar verschlüsselt) halten und automatisch auslesen. Dann kannst Du das Programm weitergeben, ohne dass jemand mit Deinen Daten Unfug treiben kann.)

### Warnung:

Du hast gesehen: Informationen wie Absendezeit und Absenderadresse haben **wir selbst** in den Mail-Text eingetragen und hätten da auch irgendwas anderes hinschreiben können. Solcher Unfug wird inzwischen von fast allen SMTP-Servern erkannt (sie kennen ja ebenfalls die Uhrzeit, die Parameter von `sendmail(absender, adressat, ...)`, sowie Deine Anmelde Daten) und als mögliche missbräuchliche Aktion mit Verweigerung, Verbindungsabbruch und/oder Strafwarteminuten belohnt.