

Natürlich, ich erstelle dir einen sehr detaillierten Lernzettel mit einfachen, ausführlichen Erklärungen und klar kommentierten Lernbeispielen. Ziel ist, dass du Schritt für Schritt verstehst, wie du E-Mails auf technische Seriosität prüfen kannst und diese Checks in deinem Python-Programm einsetzt. Alle Beispiele sind so gestaltet, dass sie beim Lernen helfen, nicht einfach zum Kopieren da sind.

Lernzettel: E-Mail-Sicherheit in Python – Schritt für Schritt

1. Ziel dieses Lernzettels

Du möchtest, dass dein Programm automatisch auf E-Mails antwortet – aber nur, wenn diese seriös sind. Deshalb prüfst du:

- Ob die E-Mail-Adresse gültig ist
 - Ob die Domain einen Mailserver hat
 - Ob die Mail von einem erlaubten Server (SPF) kommt
 - Ob die IP des Absenders einen schlechten Ruf hat
 - Ob der Inhalt verdächtige Wörter enthält
-

2. Vorbereitung: Module installieren

Damit du die Funktionen benutzen kannst, musst du diese Module installieren:

- `email-validator` für E-Mail-Validierung
- `dnspython` für DNS-Abfragen
- `pyspf` für SPF-Prüfung
- `requests` für API-Anfragen (z. B. AbuseIPDB)

Installiere sie mit diesen Befehlen im Terminal:

```
pip install email-validator dnspython pyspf requests
```

3. Schritt 1: E-Mail-Adresse technisch prüfen

Warum?

Weil es viele falsch geschriebene oder ungültige E-Mail-Adressen gibt. Bevor dein Programm antwortet, sollte es nur gültige Adressen akzeptieren.

Was wird geprüft?

- Ist die E-Mail formal korrekt? (z. B. kein zweites @-Zeichen)
- Gibt es die Domain überhaupt?
- Hat die Domain einen Mailserver?

Lernbeispiel:

```
# Wir importieren die nötigen Funktionen aus dem Modul email_validator
from email_validator import validate_email, EmailNotValidError

def pruefe_email_adresse(email_adresse):
    """
    Prüft, ob eine E-Mail-Adresse technisch gültig und zustellbar ist.

    Parameter:
    email_adresse (str): Die zu prüfende E-Mail-Adresse.

    Rückgabe:
    bool: True, wenn gültig und zustellbar, sonst False.
    """
    try:
        # validate_email prüft Syntax und Domain
        # check_deliverability=True prüft, ob Mailserver (MX-Einträge)
        existieren
        ergebnis = validate_email(email_adresse, check_deliverability=True)

        # Wenn kein Fehler auftrat, ist die Adresse gültig
        print(f"Die Adresse '{email_adresse}' ist gültig.")
        # Das Ergebnis enthält eine bereinigte Form der Adresse (z.B.
        Kleinbuchstaben)
        print(f"Bereinigte Adresse: {ergebnis.email}")

        return True
    except EmailNotValidError as fehler:
        # Wenn ein Fehler auftaucht, ist die Adresse ungültig
        print(f"Ungültige E-Mail-Adresse: {email_adresse}")
        print(f" Fehlergrund: {fehler}")
        return False

# Beispieldaufrufe (bitte eigene Testadressen verwenden):
pruefe_email_adresse("valid@example.com")
pruefe_email_adresse("ungültig@@beispiel.de")
```

Erklärung:

- Der `try`-Block versucht die Adresse zu prüfen.
- Bei Fehlern (Ausnahmen) landet das Programm im `except`-Block, der dir den Fehler anzeigt.

- So kannst du falsche Adressen aussortieren.
-

4. Schritt 2: Mailserver der Domain (MX-Einträge) prüfen

Warum?

Eine Domain kann E-Mails nur empfangen, wenn sie Mailserver eingetragen hat. Ist kein MX-Eintrag vorhanden, ist die Domain für Mailverkehr nicht zuständig.

Lernbeispiel:

```
import dns.resolver

def pruefe_mx_eintraege(domain):
    """
    Prüft, ob eine Domain mindestens einen Mailserver (MX-Eintrag) hat.

    Parameter:
    domain (str): Die Domain, z.B. 'example.com'

    Rückgabe:
    bool: True, wenn MX-Einträge vorhanden, sonst False.
    """
    try:
        # Mit resolve fragen wir die MX-Einträge der Domain ab
        antworten = dns.resolver.resolve(domain, 'MX')

        print(f"MX-Einträge für Domain '{domain}':")
        for eintrag in antworten:
            # exchange gibt den Mailserver an
            print(f"  - Mailserver: {eintrag.exchange}")

        return True
    except Exception as fehler:
        # Falls keine MX-Einträge gefunden wurden oder Fehler auftraten
        print(f"Keine MX-Einträge für Domain '{domain}' gefunden.")
        print(f" Fehler: {fehler}")
        return False

# Beispieldaufrufe:
pruefe_mx_eintraege("example.com")
pruefe_mx_eintraege("nichtexistierende-domain.tld")
```

Erklärung:

- Wenn `resolve` keine MX-Einträge findet, wird eine Ausnahme ausgelöst.
 - Du kannst Domains ohne Mailserver direkt ablehnen.
-

5. Schritt 3: SPF-Eintrag prüfen (Sender Policy Framework)

Warum?

SPF schützt davor, dass Betrüger E-Mails mit gefälschter Absenderadresse senden. Er erlaubt nur bestimmten Servern, Mails im Namen einer Domain zu versenden.

Lernbeispiel:

```
import spf

def pruefe_spf(ip_adresse, absender_email, helo_name):
    """
    Prüft, ob eine IP-Adresse laut SPF-Eintrag der Domain den Versand der Mail
    erlaubt.

    Parameter:
    ip_adresse (str): IP-Adresse des sendenden Servers.
    absender_email (str): Absenderadresse der E-Mail.
    helo_name (str): Hostname, den der sendende Server beim Mailversand nennt.

    Rückgabe:
    str: Ergebnis des SPF-Checks ('pass', 'fail', 'softfail', 'neutral' etc.)
    """
    ergebnis = spf.check2(ip_adresse, absender_email, helo_name)

    print(f"SPF-Ergebnis für IP {ip_adresse} und Absender {absender_email}:
{ergebnis[0]}")

    return ergebnis[0]

# Beispieldaufruf:
# Nehme IP, Absender und helo aus echten Mails (z.B. aus dem Header)
pruefe_spf("203.0.113.5", "user@example.com", "mail.example.com")
```

Erklärung:

- Die IP ist der Server, der die Mail gesendet hat (z.B. aus E-Mail-Header).
- Der `helo_name` ist der Hostname, den dieser Server beim Senden benutzt.
- `check2` gibt ein Tupel zurück, wobei das erste Element das Ergebnis ist.

6. Schritt 4: IP-Reputation prüfen (optional, mit AbuseIPDB)

Warum?

Manche Server sind zwar legitim, stehen aber auf schwarzen Listen wegen Missbrauch. Die IP-Reputation gibt dir ein zusätzliches Sicherheitsmaß.

Vorbereitung:

- Registriere dich bei [AbuseIPDB](#) und hole dir einen kostenlosen API-Key.

Lernbeispiel:

```
import requests

def pruefe_ip_reputation(ip_adresse, api_key):
    """
    Prüft die Reputation einer IP-Adresse über die AbuseIPDB API.

    Parameter:
    ip_adresse (str): Die IP-Adresse, die geprüft werden soll.
    api_key (str): Dein persönlicher API-Schlüssel für AbuseIPDB.

    Rückgabe:
    int: Missbrauchs-Score (0-100), höher = riskanter.
    """
    url = f"https://api.abuseipdb.com/api/v2/check?ipAddress={ip_adresse}"
    headers = {
        'Key': api_key,
        'Accept': 'application/json'
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        daten = response.json()
        score = daten['data']['abuseConfidenceScore']
        print(f"Missbrauchs-Score für IP {ip_adresse}: {score}")
        return score
    else:
        print(f"Fehler bei API-Anfrage: {response.status_code}")
        return -1

# Beispielaufruf (API-Key einsetzen):
# pruefe_ip_reputation("203.0.113.5", "DEIN_API_KEY")
```

Erklärung:

- Der Score ist ein Wert von 0 bis 100.
- Werte über 50 sind Warnsignale.
- Der API-Key ist zwingend erforderlich.

7. Schritt 5: E-Mail-Inhalt auf verdächtige Wörter prüfen

Warum?

Viele Phishing-Mails benutzen typische Formulierungen, um den Empfänger unter Druck zu setzen.

Lernbeispiel:

```
def pruefe_text_auf_verdacht(text):
    """
    Prüft, ob ein Text verdächtige Schlüsselwörter enthält.

    Parameter:
    text (str): Der Text der E-Mail.

    Gibt Warnungen aus, wenn verdächtige Wörter gefunden werden.

```

```
"""
# Liste mit typischen Phishing-Wörtern
verdachts_woerter = ["konto gesperrt", "passwort zurücksetzen", "jetzt
handeln", "dringend", "sofort klicken"]

# Um Vergleich einfacher zu machen, alles klein schreiben
text_klein = text.lower()

# Prüfen, ob eines der Wörter im Text vorkommt
for wort in verdachts_woerter:
    if wort in text_klein:
        print(f"Warnung: Verdächtiges Wort gefunden: '{wort}'")

# Beispielanwendung:
mail_text = "Sehr geehrter Nutzer, Ihr Konto wurde gesperrt. Bitte jetzt
handeln!"
pruefe_text_auf_verdacht(mail_text)
```

8. Zusammenführung und Anwendung im Programm

- Führe die Schritte der Reihe nach aus, sobald eine neue E-Mail eingeht.
- Zuerst prüfst du die Absenderadresse (Schritt 1).
- Dann die Domain (Schritt 2) und SPF (Schritt 3).
- Optional kannst du zusätzlich die IP-Reputation prüfen (Schritt 4).
- Schließlich analysierst du den Text (Schritt 5).

9. Nächste Schritte

- E-Mail-Header auslesen, um IP-Adresse und helo-Namen zu bekommen (für SPF und AbuseIPDB).
- Ergebnisbewertung, z. B. ein Punktesystem für Risiko.
- Umgang mit Ausnahmen und Fehlermeldungen in deinem Programm.

Wenn du möchtest, helfe ich dir bei der nächsten Stufe — z.B. wie du E-Mail-Header mit Python analysierst, oder wie du die einzelnen Prüfschritte in einer Klasse organisierst.

Ich hoffe, dieser Lernzettel hilft dir, den Prozess komplett und verständlich zu erfassen!