Alles klar! Hier ist dein ausführlicher, gut kommentierter Lernzettel zum Python-Standardmodul **`email`** — mit Fokus auf E-Mails lesen, analysieren und versenden (ohne Anhänge), sehr detailliert und praxisnah.

Lernzettel: Python-Modul `email`

---

1. Überblick

Das `email`-Modul ist dein Werkzeug, um E-Mails in Python **programmgesteuert** zu **erstellen, analysieren (parsen), lesen und bearbeiten**.
Es unterstützt die verschiedenen Teile einer E-Mail: Header, Body, MIME-Typen, Encodings usw.

Wichtig: Das Modul allein kann keine Mails senden oder empfangen — dafür nutzt du zusätzlich `smtplib` (Senden) und `imaplib` oder `poplib` (Empfangen). Aber `email` ist das Kernmodul zum Bearbeiten der E-Mail-Struktur.

---

2. Grundlegende Begriffe

- **Header**: Meta-Informationen wie Absender, Empfänger, Betreff, Datum (`From`, `To`, `Subject`...)
- **Body**: Textinhalt der Nachricht (meist als Plaintext oder HTML)
- **MIME**: Standard für E-Mail-Formate, um Text, HTML, Anhänge zu strukturieren
- **Message-Objekt**: Repräsentation einer E-Mail in Python (Klasse `email.message.EmailMessage` oder `email.message.Message`)

---

3. E-Mail erstellen (ohne Anhang)

```
from email.message import EmailMessage

# Neues EmailMessage-Objekt anlegen
msg = EmailMessage()

# Header setzen
msg['From'] = "absender@example.com"
msg['To'] = "empfaenger@example.com"
msg['Subject'] = "Testnachricht"

# Body setzen (Plaintext)
msg.set_content("Hallo,\nDies ist eine automatisch generierte Nachricht.\nBeste
Grüße!")

# Ausgabe der fertigen E-Mail (als String)
print(msg.as_string())
```

Erklärung:

- `EmailMessage()` erzeugt eine neue Mail.
- Header werden über Dictionary-Syntax gesetzt.
- `set_content()` fügt den Textkörper ein.
- `as_string()` wandelt die Mail in das Format um, das man z.B. per SMTP senden kann.

---

4. E-Mail parsen (aus rohem Text)

```
from email import message_from_string

raw_email = """\
From: absender@example.com
To: empfaenger@example.com
Subject: Testnachricht

Hallo,
Dies ist eine automatisch generierte Nachricht.
Beste Grüße!
"""

# Parse die rohe Email in ein Message-Objekt
msg = message_from_string(raw_email)

# Header auslesen
print("Von:", msg['From'])
print("An:", msg['To'])
print("Betreff:", msg['Subject'])

# Body auslesen
if msg.is_multipart():
    # Multipart-Mails enthalten mehrere Teile, z.B. Text + HTML
    for part in msg.walk():
        # Nur Textteile ausgeben
        if part.get_content_type() == "text/plain":
            print("Body:",
part.get_payload(decode=True).decode(part.get_content_charset()))
else:
    # Einfache Mail: Body direkt holen
    print("Body:", msg.get_payload())
```

Erklärung:

- `message_from_string()` wandelt Rohtext in ein E-Mail-Objekt um.
- Header liest du per `msg['Headername']`.
- Bei multipart-Mails (z.B. mit HTML und Text) durchläufst du die Teile mit `.walk()`.
- `.get_payload(decode=True)` gibt den Inhalt (Body) als Bytes, das man dann decodiert.

---

5. Body mit mehreren Formaten lesen (Plaintext & HTML)

Viele Mails haben neben Plaintext auch HTML als Body. Das erkennst du mit `is_multipart()`.

```
if msg.is_multipart():
    for part in msg.walk():
        ctype = part.get_content_type()
        if ctype == "text/plain":
            text =
part.get_payload(decode=True).decode(part.get_content_charset())
            print("Plaintext:", text)
        elif ctype == "text/html":
            html =
part.get_payload(decode=True).decode(part.get_content_charset())
            print("HTML:", html)
```

---

6. E-Mail bearbeiten und ändern (z.B. Antwort erstellen)

```
from email.message import EmailMessage

# Ausgegangene Mail parsen (Beispiel)
original_msg = message_from_string(raw_email)

# Neue Antwort erstellen
reply = EmailMessage()

# Antwort-Header setzen
reply['From'] = "deinemail@example.com"
reply['To'] = original_msg['From']  # Antwort an ursprünglichen Absender
reply['Subject'] = "Re: " + original_msg['Subject']

# Inhalt der Antwort
reply.set_content("Danke für deine Nachricht! Ich melde mich bald zurück.")

print(reply.as_string())
```

---

7. Encoding und Zeichensatz

E-Mails können unterschiedliche Zeichensätze haben (z.B. UTF-8). Das Modul kümmert sich meist darum, aber du kannst es auch selbst setzen:

```
msg.set_content("Nachricht mit Umlauten äöü", charset='utf-8')
```

Wenn du Text aus einem Payload liest, frage den Zeichensatz ab:

```
charset = part.get_content_charset() or 'utf-8'
text = part.get_payload(decode=True).decode(charset)
```

---

8. Tipps für deinen Use Case: E-Mails prüfen & antworten

- **E-Mail abrufen:** (z.B. via `imaplib`) -> Rohtext der Mail holen -> mit `email.message_from_bytes()` in ein Message-Objekt parsen

- **E-Mail auswerten:** Header (From, Subject, etc.) und Body analysieren
- **Antwort generieren:** neues `EmailMessage`-Objekt anlegen, Absender auf Antwortadresse setzen, Betreff mit "Re:" erweitern, Text einfügen
- **Mail senden:** Mit `smtplib` die Antwort versenden

---

9. Kurzes Beispiel, wie man eine empfangene Mail (Bytes) parst

```
from email import message_from_bytes

# Beispiel: E-Mail als Bytes (z.B. von IMAP-Server)
raw_bytes = b"From: a@example.com\nTo: b@example.com\nSubject: Hallo\n\nText"

msg = message_from_bytes(raw_bytes)

print(msg['From'])
print(msg.get_payload())
```

---

10. Zusammenfassung

- `email` ist dein Parser und Composer für E-Mails in Python.
- Header über Dictionary-Syntax lesen und schreiben (`msg['Subject'] = "..."`).
- Body mit `.set_content()` schreiben, mit `.get_payload()` lesen.
- Multipart-Mails brauchen `.is_multipart()` + `.walk()` zum Durchlaufen.
- Immer auf Zeichensatz beim Dekodieren achten (`get_content_charset()`).
- Zum Versenden / Empfangen nutzt du `smtplib` / `imaplib` zusätzlich.

Falls du möchtest, kann ich dir auch Beispielcodes und Erklärungen zu `smtplib` und `imaplib` machen, damit du das Gesamtbild bekommst!

Möchtest du das?