Klar! Hier kommt dein **detaillierter Lernzettel zu `bcrypt` in Python**. Damit kannst du das sichere Hashen von Passwörtern verstehen und direkt in deinem Programm (z.B. ReMailD) verwenden.

---

# Lernzettel: Sicheres Passwort-Handling mit `bcrypt` in Python

---

## 🔎 Was ist `bcrypt`?

- `bcrypt` ist ein **sicherer Algorithmus** zum Erstellen von **Passwort-Hashes**.
- Ein Hash ist eine **einweg**-Funktion:
  Man kann aus dem Hash nicht das ursprüngliche Passwort zurückgewinnen.
- Wichtig für sichere Speicherung von Passwörtern:
  **Nie Passwörter als Klartext speichern!**

---

## 🛠️ Installation

```
pip install bcrypt
```

---

## 🧩 Die wichtigsten Funktionen von `bcrypt`

| Funktion | Beschreibung | Verwendung im Programm |
|---|---|---|
| `bcrypt.hashpw()` | Erzeugt einen sicheren Hash aus dem Klartext-Passwort und einem Salt | Passwort beim Setup oder bei Passwortänderung hashen |
| `bcrypt.gensalt()` | Generiert einen zufälligen Salt-Wert für den Hash | Bei jedem Hashvorgang verwenden, erhöht die Sicherheit |
| `bcrypt.checkpw()` | Prüft, ob ein eingegebenes Passwort zum gespeicherten Hash passt | Login oder Passwort-Abfrage validieren |

---

## 📖 Detaillierte Erläuterung mit Dummy-Code

---

### 1️⃣ Passwort sicher speichern (Hash erzeugen)

```
import bcrypt

# 1.1. Klartext-Passwort (vom User eingegeben)
klartext_passwort = b"mein_super_geheimes_passwort"
```

```
# 1.2. Salt erzeugen (zufällige Daten, die den Hash einzigartig machen)
salt = bcrypt.gensalt()

# 1.3. Passwort mit Salt hashen (erzeugt sicheren Hash)
hashed_passwort = bcrypt.hashpw(klartext_passwort, salt)

# 1.4. Hash speichern (z. B. in einer Datei oder Datenbank)
print("Speichere Hash:", hashed_passwort)
```

**Wichtig:**

- Das Passwort muss als `bytes` (also z.B. `b"string"`) übergeben werden!
- `bcrypt.gensalt()` sorgt dafür, dass gleiche Passwörter verschiedene Hashes bekommen.
- Speichere nur den Hash, **nicht das Klartext-Passwort!**

---

## 2️⃣ Passwort prüfen (Login oder Passwort-Abfrage)

```
import bcrypt

# 2.1. Klartext-Passwort, das der Benutzer gerade eingibt
eingabe_passwort = b"mein_super_geheimes_passwort"

# 2.2. Hash, den du beim Setup gespeichert hast (z. B. aus Datei gelesen)
gespeicherter_hash = b"$2b$12$..."

# 2.3. Prüfen, ob das eingegebene Passwort zum gespeicherten Hash passt
if bcrypt.checkpw(eingabe_passwort, gespeicherter_hash):
    print("Login erfolgreich! Passwort stimmt.")
else:
    print("Login fehlgeschlagen! Passwort falsch.")
```

**Wichtig:**

- `checkpw()` gibt `True` zurück, wenn das Passwort passt.
- Du brauchst keinen eigenen Vergleich von Hashes oder Klartext!

---

## 3️⃣ Praktischer Ablauf im Programm

```
def setup_passwort():
    # 1. Benutzer gibt Passwort ein (in bytes)
    pw = input("Bitte Admin-Passwort festlegen: ").encode()

    # 2. Passwort hashen
    salt = bcrypt.gensalt()
    hashed = bcrypt.hashpw(pw, salt)

    # 3. Hash sicher speichern (z.B. in Datei)
    with open("admin_passwort.hash", "wb") as f:
        f.write(hashed)
    print("Passwort gespeichert!")

def login():
    # 1. Benutzer gibt Passwort ein
```

```
    eingabe = input("Admin-Passwort eingeben: ").encode()

    # 2. Hash aus Datei laden
    with open("admin_passwort.hash", "rb") as f:
        gespeicherter_hash = f.read()

    # 3. Passwort prüfen
    if bcrypt.checkpw(eingabe, gespeicherter_hash):
        print("Login OK")
        return True
    else:
        print("Falsches Passwort")
        return False
```

---

## 💡 Tipps und Best Practices

- **Nie Klartext-Passwörter speichern oder protokollieren!**
- Verwende `bcrypt` immer mit `gensalt()`.
- Speichere den Hash an einem sicheren Ort, z.B. eine versteckte Datei mit eingeschränkten Zugriffsrechten.
- Für CLI-Passworteingabe nutze `getpass.getpass()` statt `input()`, damit das Passwort nicht angezeigt wird.
- Hashes können sicher in `.json` oder `.txt` als `bytes` (Base64-kodiert) gespeichert werden.

---

## ✍️ Beispiel: Hash in JSON speichern

```
import json
import bcrypt
import base64

# Passwort hashen
pw = b"mein_passwort"
hashed = bcrypt.hashpw(pw, bcrypt.gensalt())

# Bytes zu String für JSON (Base64)
hashed_str = base64.b64encode(hashed).decode()

# In JSON speichern
with open("passwort.json", "w") as f:
    json.dump({"admin_hash": hashed_str}, f)

# Zum Laden
with open("passwort.json") as f:
    data = json.load(f)
    loaded_hash = base64.b64decode(data["admin_hash"].encode())

# Prüfen
eingabe = b"mein_passwort"
if bcrypt.checkpw(eingabe, loaded_hash):
    print("Passwort OK")
```

---

## 🔥 Zusammenfassung

| Schritt | Funktion | Was du tun musst |
|---|---|---|
| Passwort absichern | `bcrypt.hashpw()` + `gensalt()` | Passwort hashen und speichern |
| Passwort prüfen | `bcrypt.checkpw()` | Eingabe mit gespeichertem Hash vergleichen |
| Hash speichern/laden | Datei/DB + ggf. Base64 | Hash in Datei sicher ablegen und lesen |
| Eingabe im CLI | `getpass.getpass()` | Passwort sicher eingeben lassen (unsichtbar) |

---

Mit diesem Lernzettel kannst du jetzt **selbstständig ein sicheres Passwortsystem mit `bcrypt`** in deinem Python-Programm bauen und Fehler vermeiden.

Falls du magst, helfe ich dir gern noch mit einer konkreten Integration in dein ReMailD-Projekt!