Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Hannes Saffrich
Wintersemester 2020

Universität Freiburg
Institut für Informatik

Übungsblatt 6

Abgabe: Montag, 14.12.2020, 9:00 Uhr morgens, über git[1]

Hinweis

Aufgabenteile werden mit **0 Punkten** bewertet wenn:

- Dateien und Definitionen nicht so benannt sind, wie im Aufgabentext gefordert;
- Dateien falsche Formate haben, z.B. PDF statt plaintext; oder
- Pythonskripte wegen eines Syntaxfehlers nicht ausführbar sind.

Es gibt **Punktabzug** wenn:

- Funktionen keine oder falsche Typannotationen aufweisen. Ausnahme: Bei Funktionen, die stets `None` zurückgeben, kann der Rückgabetyp weggelassen werden.

Gruppenaufgaben müssen von allen Mitgliedern abgegeben werden und in der ersten Zeile müssen die Mitglieder in einem Kommentar vermerkt werden, z.B:

```
# Gruppe: xy123, yz56, zx934
```

Hinweis

Hier ist eine Übersicht der bisherigen Typannotationen:

```
from typing import Union, Optional
x: int              = 42
x: float            = 42.0
x: complex          = 42.0 + 23.0 * j
x: bool             = True
x: str              = 'foo'
x: type(None)       = None
x: list[int]        = [1, 2, 3]
x: list[list[int]] = [[1, 2, 3], [4, 5]]
x: MyClass          = MyClass(...)  # Angenommen MyClass wurde definiert
x: Union[int,bool] = 42
x: Union[int,bool] = True
x: Optional[int]   = None           # Kurzform für Union[int, type(None)]
```

Für komplexere `Union`s empfiehlt es sich zur Lesbarkeit Typ-Aliase zu definieren:

```
MyType = Union[int, bool, list[int], MyClass, type(None)]
def combine(x: MyType, y: MyType) -> MyType:
    # ...
```

[1] `https://inpro.informatik.uni-freiburg.de/`

Aufgabe 6.1 (Supermarkt; Datei: `supermarket.py`; Punkte: 2+2+2+2)

In dieser Aufgabe sollen Sie die Lagerbestände eines Supermarkts modellieren.

Der Lagerbestand einer Ware ist entweder verzehrbar oder nicht-verzehrbar.

Der Lagerbestand für eine nicht-verzehrbare Ware, `OtherStock`, besteht aus dem Namen der Ware `name`, der Anzahl der gelagerten Waren `units` und dem Stückpreis in Cents `price_per_unit`.

Der Lagerbestand für eine verzehrbare Ware, `FoodStock`, besteht zusätzlich aus einem Mindesthaltbarkeitsdatum `expiration_date`.

(a) Erstellen Sie zwei Datenklassen, `FoodStock` und `OtherStock`, für verzehrbare und nicht-verzehrbare Lagerbestände[2].

Modellieren Sie ein Datum dabei als einen String, der für den 14. Februar 2021 das Format `"2021-02-14"` hat[3]. Dieses Format erlaubt es einfach die lexikographische Ordnung zu verwenden, um herauszufinden ob eines von zwei Daten weiter in der Zukunft liegt wie das andere.

Definieren Sie für allgemeine Lagerbestände einen `Union`-Typ `Stock`, der Werte beschreibt, die entweder eine Instanz von `FoodStock` oder eine Instanz von `OtherStock` sind[4].

(b) Schreiben Sie eine Funktion `is_expired`, die einen Lagerbestand und ein Datum als Argumente nimmt und zurückgibt ob der Lagerbestand zu diesem Datum abgelaufen ist. Verzehrbare Waren gelten dabei ab einem Tag nach ihrem Mindesthaltbarkeitsdatum als abgelaufen. Nicht-verzehrbare Waren gelten nie als abgelaufen.

(c) Schreiben Sie eine Funktion `get_expired`, die eine Liste von Lagerbeständen und ein Datum als Argumente nimmt und eine Liste derjenigen Lagerbestände zurückgibt, die zu dem Datum abgelaufen sind.

(d) Schreiben Sie eine Funktion `buy`, die einen Lagerbestand und eine Stückzahl als Argumente nimmt, den Lagerbestand um die Stückzahl der Waren verringert, und die Anzahl der gekauften Waren zurückgibt.

Der Lagerbestand soll dabei nie weniger als 0 Waren enthalten, d.h. wenn mehr Waren gefordert werden, wie im Lagerbestand verfügbar sind, so sollen nur die verfügbaren Waren verbucht werden.

[2] Verwenden Sie keine Vererbung. Vererbung wird erst zu einem späteren Zeitpunkt behandelt und ist nicht das Konzept welches hier geübt werden soll.

[3] Hintergrundinfo: dieses Format ist eine Variante des ISO 8601 Standards.

[4] Siehe Hinweis auf Seite 1.

Aufgabe 6.2 (Gruppenaufgabe: Mail Server; Datei: `mail.py`; Punkte: 2+2+2+2+2)

In dieser Aufgabe sollen Sie den Versand von E-Mails zwischen mehreren Mail-Server modellieren.

Eine Email-Adresse `MailAddress` besteht aus einem Namen `name` und einer Domain `domain`.

Eine Email `Mail` besteht aus den Email-Adressen von Absender `sender` und Empfänger `receiver`, einem Betreff `subject` und dem Nachrichtenkörper `body`.

Ein Email-Account `MailAccount` besteht aus einem Namen `name`, und jeweils einer Listen von Emails für den Posteingang `inbox` und den Postausgang `outbox`.

Ein Email-Server `MailServer` besteht aus einer Domain `domain` und einer Liste von E-Mail-Accounts `accounts`.

(a) Modellieren Sie das oben beschriebene Szenario mit 4 Datenklassen.

(b) Schreiben Sie Funktionen `show_mail_address`, `show_mail`, `show_mail_account`, und `show_mail_server`, die eine Instanz der jeweiligen Datenklassen als Argument nehmen, zu einem lesbaren String umwandeln und diesen zurückgegeben. Hierbei sollen alle Felder der Datenklasse im String dargestellt werden.

Die Funktionen `show_mail_address` und `show_mail` sollen sich dabei *exakt* wie in folgendem Beispiel verhalten:

```
>>> print(show_mail_address(MailAddress("me", "mydomain.com")))
me@mydomain.com
>>> mail = Mail(
        MailAddress("me", "mydomain.com"),
        MailAddress("you", "yourdomain.com"),
        "Important!!1",
        "Hi you,\n\nmaybe it's not that important after all...")
>>> print(show_mail(mail))
From: me@mydomain.com
To: you@yourdomain.com
Subject: Important!!1

Hi you,

maybe it's not that important after all...
```

(c) Schreiben Sie eine Funktion `find_server`, die eine Domain und eine Liste von E-Mail-Server als Argumente nimmt und die Liste nach einem E-Mail-Server durchsucht der die gefragte Domain hat. Wird solch ein Server gefunden soll dieser zurückgegeben werden, ansonsten soll als Alternative `None` zurückgegeben werden. Denken Sie daran die Alternative im Rückgabetyp zu berücksichtigen.
Schreiben Sie eine Funktion `find_account`, die analog zu `find_server` einen Mail-Server nach einem Account mit einem bestimmten Namen durchsucht.

(d) Schreiben Sie eine Funktion `deliver_mail`, die eine E-Mail und eine Liste von E-Mail-Server als Argumente nimmt, und versucht die E-Mail ihrem Empfänger zuzustellen. Wird der Empfänger gefunden, so soll die E-Mail im Posteingang des Empfänger-Accounts hinzugefügt werden und `True` zurückgegeben werden. Wird der Empfänger nicht gefunden, soll `False` zurückgegeben werden.

(e) Schreiben Sie eine Funktion `deliver_all_mail`, die eine Liste von E-Mail-Server als Argument nimmt, und die E-Mails in den Postausgängen aller vorhandener Accounts mit `deliver_mail` zuzustellen. Nach der Zustellung sollen die Postausgänge leer sein. Die Zustellung soll dabei nur erfolgen, wenn der Absenderadresse authentisch ist, d.h. wenn der Name der Adresse mit dem Accountnamen übereinstimmt und die Domain der Adresse mit der Server-Domain.

Aufgabe 6.3 (Erfahrungen; Datei: `erfahrungen.txt`; 2 Punkte)

Notieren Sie hier Ihre Erfahrungen mit diesem Übungsblatt (benötigter Zeitaufwand, Probleme, Bezug zur Vorlesung, Interessantes, etc.).

Schreiben Sie den groben Zeitaufwand, den Sie für das *gesamte* Blatt benötigt haben, bitte wie folgt in die erste Zeile:

```
Zeitaufwand: 3.5h

[... Freitext, wie bisher ...]
```

Wenn sich genug Leute daran halten, dann können wir ein Pythonskript schreiben, welches automatisiert die durchschnittliche Bearbeitungszeit berechnet ;-)