Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Hannes Saffrich, Simon Ging
Wintersemester 2021

Universität Freiburg
Institut für Informatik

Übungsblatt 6

Abgabe: Montag, 29.11.2021, 9:00 Uhr morgens

Typannotationen

Sie müssen bei jeder Funktionsdefinition korrekte Typannotationen für die Argumente und den Rückgabewert angeben. Bei Verstößen gibt es Punktabzug.

Hinweis zu Gruppenaufgaben

Da die Gruppenaufgaben noch immer von einigen Leuten falsch abgegeben wurden, hier eine Wiederholung.

Schreiben Sie bei Gruppenaufgaben immer in die erste Zeile der Abgabedatei einen Kommentar mit den RZ-Accounts Ihrer Gruppe (auch wenn Sie alleine abgeben). Beispiel:

```
# ab123, xy234
```

Die Gruppenaufgabe muss von allen Gruppenteilnehmer hochgeladen werden und sollte identisch sein.

Abgaben die diese Kriterien nicht erfüllen werden mit 0 Punkten bewertet. Dies gilt auch für alle folgenden Übungsblätter.

Hinweis

Hier ist eine Übersicht der bisherigen Typannotationen:

```
from typing import Optional
x: int             = 42
x: float           = 42.0
x: complex         = 42.0 + 23.0 * 1j
x: bool            = True
x: str             = 'foo'
x: 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: int | bool      = 42
x: int | bool      = True
x: Optional[int]   = None          # Kurzform für int | None
```

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

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

```
# ...
```

Die Anzahl Elemente in `list` ist nicht festgelegt. Für eine festgelegte Anzahl von Elementen in einer Sequenz empfiehlt sich `tuple`:

```
x: tuple[int, str] = (42, 'foo')
x: list[tuple[int, str]] = [(42, 'foo'), (43, 'bar')]
```

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.

Sie können die gesamte Funktionalität mit dem beigefügten Skript `test_supermarket.py` testen. Das Testskript soll nicht verändert werden.

(a) Erstellen Sie zunächst zwei Datenklassen `Food` und `NonFood`, wobei `NonFood` eine leere Klasse ist und `Food` ein Mindesthaltbarkeitsdatum enthält. Modellieren Sie das Datum dabei als einen String, der für den 14. Februar 2021 das Format `"2021-02-14"` hat[1]. Dieses Format erlaubt es einfach die lexikographische Ordnung zu verwenden, um herauszufinden ob eines von zwei Daten weiter in der Zukunft liegt als das andere.

Erstellen Sie nun die Datenklasse `Stock` mit dem Namen der Ware `name`, der Anzahl der gelagerten Waren `units`, dem Stückpreis in Cents `price_per_unit` sowie dem Feld `kind`. Das Feld `kind` soll eine Instanz der Datenklassen `Food` oder `NonFood` enthalten, definieren Sie es also als einen `Union`-Typ über die beiden Datenklassen.

(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. Tipp: Verwenden Sie pattern matching, um den Typ des Feldes `kind` zu prüfen.

(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.

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

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`.

Sie können die gesamte Funktionalität mit dem beigefügten Skript `test_mail.py` testen. Das Testskript soll nicht verändert werden.

(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-Servern 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 die 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; 2 Punkte; Datei: `NOTES.md`)

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

Editieren Sie hierzu die Datei `NOTES.md` im Abgabeordner dieses Übungsblattes auf unserer Webplatform. Halten Sie sich an das dort vorgegebene Format, da wir den Zeitbedarf mit einem Python-Skript automatisch statistisch auswerten. Die Zeitangabe `3.5 h` steht dabei für 3 Stunden 30 Minuten.