

Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Hannes Saffrich, Michael Uhl
Wintersemester 2022

Universität Freiburg
Institut für Informatik

Übungsblatt 6

Abgabe: Montag, 28.11.2022, 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

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 # Alternativ für int | None
```

Für komplexere Unions 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 der Elemente in `list` ist nicht festgelegt. Für eine festgelegte Anzahl von Elementen in einer Sequenz empfiehlt sich der Typ `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" hat¹. Dieses Format erlaubt es, die lexikographische Ordnung zu verwenden, um herauszufinden ob eines von zwei Daten weiter in der Zukunft liegt als das andere.

Erstellen Sie die Datenklasse `Stock` mit folgenden Feldern: Namen der Ware `name`, Anzahl der gelagerten Waren `units`, Stückpreis in Cents `price_per_unit` sowie `kind`. Das Feld `kind` soll eine Instanz der Datenklassen `Food` oder `NonFood` enthalten, definieren Sie es also als `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-verzehrbar Waren gelten nie als abgelaufen. Verwenden Sie Pattern Matching, um den Inhalt 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 als im Lagerbestand verfügbar sind, so sollen nur die verfügbaren Waren verbucht werden.

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

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

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

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

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

Ein E-Mail-Account `MailAccount` besteht aus einem Namen `name` und jeweils einer Liste von E-Mails für den Posteingang `inbox` und den Postausgang `outbox`.

Ein Mailserver `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 zurückgeben. Hierbei sollen alle Felder der Datenklasse im String dargestellt werden.

Die Funktionen `show_mail_address` und `show_mail` sollen sich dabei *exakt* wie im folgenden Beispiel verhalten

```
>>> print(show_mail_address(MailAddress("me", "mydomain.com")))
me@mydomain.com
>>> mail = Mail(
    MailAddress("me", "mydomain.com"),
    MailAddress("you", "yourdomain.com"),
    "Important!!!",
    "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!!!
```

Hi you,

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

- (c) Schreiben Sie eine Funktion `find_server`, die eine Domain und eine Liste von Mailservern als Argumente nimmt und die Liste nach einem Mailserver durchsucht, der die gefragte Domain hat. Wird solch ein Server gefunden, so soll dieser zurückgegeben werden, ansonsten soll als Alternative `None` zurückgegeben werden. Denken Sie daran, die Alternative im Rückgabebetyp zu berücksichtigen. Schreiben Sie eine Funktion `find_account`, die analog zu `find_server` einen Mailserver nach einem Account mit einem bestimmten Namen durchsucht.
- (d) Schreiben Sie eine Funktion `deliver_mail`, die eine E-Mail und eine Liste von Mailservern 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 Mailservern als Argument nimmt, und versucht, die E-Mails in den Postausgängen aller Accounts mit `deliver_mail` zuzustellen. Die Zustellung wird nur versucht, wenn die Absenderadresse authentisch ist, d.h. wenn der Name der Adresse mit dem Accountnamen übereinstimmt und die Domain der Adresse mit der Domain des Servers. E-Mails, deren Absenderadresse nicht authentisch ist, werden gelöscht. Nach erfolgreicher Zustellung wird die E-Mail aus dem Postausgang entfernt. Andernfalls verbleibt sie im Postausgang.

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 Webplattform. Halten Sie sich an das dort vorgegebene Format, da wir den Zeitbedarf mit einem Python-Skript automatisch statistisch auswerten. Die Zeitanzeige 3.5 h steht dabei für 3 Stunden 30 Minuten.

Bei Gruppenaufgaben müssen alle Gruppenmitglieder die Abgabe hochladen. Ferner müssen Sie die RZ-Accounts aller Gruppenmitglieder angeben, sodass wir Sie bei der Korrektur zuordnen können und unsere Tutoren die Aufgabe nicht mehrfach korrigieren müssen. Fügen Sie hierzu in der `NOTES.md`-Datei unter der Zeile für den Zeitbedarf eine weitere Zeile hinzu die exakt das folgende Format hat:

Gruppe: `xy123, xy234, xy345`

Hierbei stehen `xy123`, `xy234`, und `xy345` für die RZ-Accounts der Gruppenmitglieder. Der Buildserver überprüft ebenfalls, ob Sie das Format korrekt angegeben haben. Prüfen Sie, ob der Buildserver mit Ihrer Abgabe zufrieden ist, so wie es im Video zur Lehrplattform gezeigt wurde.