Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Tim Schulte
Wintersemester 2018/2019

Universität Freiburg
Institut für Informatik

Übungsblatt 13

Abgabe: Dienstag, 5.2.2019, 20:00 Uhr

Wichtiger Hinweis: Zur Bearbeitung der Übungsaufgaben legen Sie bitte ein neues Unterverzeichnis `sheet13` im Wurzelverzeichnis Ihrer Arbeitskopie des SVN-Repositories an. Ihre Lösungen werden dann in Dateien in diesem Unterverzeichnis erwartet.

Aufgabe 13.1 (Exceptions; `suppress.py`; Punkte: 4)

Schreiben Sie eine Funktion `suppress(f, ignore: tuple)`, welche eine parameterlose Funktion `f` und ein Tuple an Exceptions `ignore` als Argumente erhält und eine neue Funktion `g` zurückgibt. `g` soll sich dabei identisch zu `f` verhalten, solange beim Aufruf keine Exception aus `ignore` auftritt. Tritt während des Aufrufs `f()` eine solche Exception auf, so soll diese beim Aufruf `g()` ignoriert und `None` zurückgeben werden. Beispiel:

```
>>> from functools import partial
>>> def foo(n: int) -> int:
...     return 35 // n
...
>>> assert suppress(partial(foo, 1), ())() == 35 == foo(1)
>>> suppress(partial(foo, 0), (ZeroDivisionError))()
>>> suppress(partial(foo, 0), ())()
Traceback (most recent call last):
  ...
  File "suppress.py", line 26, in <lambda>
    suppress(lambda: 3 / 0, ())()
ZeroDivisionError: division by zero
```

Hinweis: In Python können Funktionen auch innerhalb von Funktionen definiert werden. Beispiel:

```
>>> def foo(s):
...     def bar():
...         print("hi", s)
...     return bar
...
>>> b = foo("dude")
>>> b()
hi dude
```

Im Beispiel ist die Variable s in der Closure von bar. Sie kann also, in der Funktionsdefinition von bar, ganz normal verwendet werden.

Aufgabe 13.2 (Comprehensions; Datei: `comprehensions.py`; Punkte: 3+3)

(a) Ein pythagoreisches Tripel (x, y, z) besteht aus drei natürlichen Zahlen x, y und z, so dass $x^2+y^2 = z^2$. Schreiben Sie eine Funktion `pythagorean_triples(n: int) -> list`, welche alle pythagoreischen Tripel mit Hilfe von List-Comprehensions berechnet und als Liste zurückgibt, so dass $x \leq n$, $y \leq n$ und $z \leq n$.

(b) Schreiben Sie eine Funktion `cookable(xs: list) -> dict`, welche eine Liste `xs` an Zutaten (jede Zutat ist ein String) als Argument erhält und alle Rezepte aus einem Dictionary `recipes`, welche mit den gegebenen Zutaten kochbar sind, als Dictionary zurückgibt. `recipes` ist wie folgt definiert:

```
recipes = {
    "Sushi":             ["Fisch", "Reis", "Nori"],
    "Sashimi":           ["Fisch", "Reis"],
    "Pfannkuchen":       ["Mehl", "Ei", "Milch"],
    "Burger":            ["Brötchen", "Rind"],
    "Burger TS":         ["Brötchen", "Rind", "Tomate", "Salat"],
    "Cheese Burger":     ["Brötchen", "Rind", "Tomate", "Käse"],
    "Gemischter Salat":  ["Salat", "Tomate", "Gurke"]
}
```

Achtung: Ihre Funktionsdefinition soll außer einer `return`-Anweisung keine weiteren Zeilen enthalten. Innerhalb des `return`-Statements dürfen/sollen allerdings ein oder mehrere (List-/Dict-/Generator-)Comprehensions benutzt werden. Beispiele:

```
>>> cookable(["Brötchen", "Tomate", "Gurke", "Salat", "Rind", "Brötchen"])
{'Burger': ['Brötchen', 'Rind'],
 'Burger TS': ['Brötchen', 'Rind', 'Tomate', 'Salat'],
 'Gemischter Salat': ['Salat', 'Tomate', 'Gurke']}
>>> cookable(["Fisch", "Reis", "Tomate"])
{'Sashimi': ['Fisch', 'Reis']}
```

Aufgabe 13.3 (Church-Numerale; Datei: `church.py`; Punkte: 2+2+2+2)

Mittels *Church-Numeralen* lassen sich natürliche Zahlen als Funktionen höherer Ordnung repräsentieren. Die Idee hierbei ist, dass eine natürliche Zahl n als eine Funktion dargestellt wird, die jeder (einstelligen) Funktion s die n-fache Hintereinanderausführung von s zuordnet. Bei sinnvoller Wahl der Funktion s und eines Nullelements ergibt die Anwendung des Church-Numerals, das die Zahl n repräsentiert, auch eine Darstellung der Zahl n. Beispiel:

```
>>> zer = lambda s: lambda z: z
>>> one = lambda s: lambda z: s(z)
>>> two = lambda s: lambda z: s(s(z))
>>> def toint(chn):
...     """Convert Church numeral to int."""
...     return chn(lambda x: x+1)(0)
...
>>> assert toint(zer) == 0
>>> assert toint(one) == 1
>>> assert toint(two) == 2
>>> def totally(chn):
...     """Convert Church numeral to tally."""
...     return chn(lambda x: x+"|")("")
...
>>> assert totally(zer) == ""
>>> assert totally(one) == "|"
>>> assert totally(two) == "||"
```

(a) Definieren Sie eine Funktion `succ(chn)`, die zu einem gegebenen Church-Numeral `chn`, die die natürliche Zahl n repräsentiert, ein Church-Numeral zurückgibt, das die Zahl $n + 1$ repräsentiert.

(b) Definieren Sie einen Generator `church(n)`, der zu einer gegebenen natürlichen Zahl `n` die Church-Numerale zur Repräsentation von 0 bis `n`-1 in aufsteigender Reihenfolge zurückgibt.

(c) Definieren Sie eine Liste `churches`, die alle Church-Numerale zur Repräsentation der Zahlen 0 bis 99 in *absteigender* Reihenfolge enthält.

(d) Schreiben Sie für die Teilaufgaben (a) und (b) jeweils eine sinnvolle, pytest-kompatible Testfunktion. Sie dürfen dazu auf die Konvertierungsfunktionen `toint` bzw. `totally` zurückgreifen.

Aufgabe 13.4 (Erfahrungen; Datei: `erfahrungen.txt`; Punkte: 2)

Legen Sie im Unterverzeichnis `sheet13` eine Textdatei `erfahrungen.txt` an. Notieren Sie in dieser Datei kurz Ihre Erfahrungen beim Bearbeiten der Übungsaufgaben (Probleme, Bezug zur Vorlesung, Interessantes, benötigter Zeitaufwand, etc.).