

Probeklausur

Objektorientierung und Funktionale Programmierung — Funktionale Programmierung mit Python

Bearbeitungszeit: 60 Minuten	Gesamtpunktzahl: 49 Punkte (+ 4 Bonuspunkte)
Hilfsmittel: keine	

Name / Matrikelnummer:

Hinweise: Schreiben Sie Python-Code leserlich und mit korrekter Einrückung. Sofern nicht anders angegeben, dürfen Sie annehmen, dass `from functools import reduce` bereits ausgeführt wurde. Kleinere Syntaxfehler führen nicht automatisch zu Punktabzug, falsche Logik schon.

Aufgabe 1: Kurzfragen (8 Punkte)

a) Nennen Sie die zwei Bedingungen, die eine Funktion erfüllen muss, um eine *pure function* zu sein. (2 P)

b) Ordnen Sie die folgenden Python-Typen in *mutable* (veränderbar) und *immutable* (unveränderbar) ein: `int`, `list`, `tuple`, `dict`, `string`, `set`, `float`, `bool` (2 P)

c) Worauf muss bei jeder rekursiven Funktion zwingend geachtet werden, und was passiert intern bei rekursiven Aufrufen (Stichwort: welche Struktur entsteht)? (2 P)

d) Was gibt `map(f, liste)` zurück — eine Liste oder etwas anderes? (2 P)

Aufgabe 2: Code-Analyse — Geltungsbereiche und Mutability (10 Punkte)

Geben Sie für jedes Programm an, was auf der Kommandozeile ausgegeben wird oder ob ein Fehler auftritt. Begründen Sie Ihre Antwort jeweils in einem Satz. (je 2 P)

a)

```
liste = [1, 2, 3]
def f():
    liste[0] = liste[0] + 10
f()
print(liste)
```

b)

```
a = [1]
def g():
    a = [5]
    a.append(7)
g()
print(a)
```

c)

```
x = 3
def h():
    global x
    x = x * 2
h()
print(x)
```

d)

```
n = 1
def k():
    n = n + 1
    return n
print(k())
```

e)

```
t = (1, 2)
def m(t):
    t = t + (3,)
    return t
s = m(t)
print(t, s)
```

Aufgabe 3: Programmieren mit map, filter und reduce (11 Punkte)

Verwenden Sie in dieser Aufgabe **keine** for- oder while-Schleifen. Nutzen Sie lambda-Ausdrücke.

a) Gegeben ist `zahlen = [1, 2, 3, 4, 5]`. Erzeugen Sie mit `map` in einer Zeile die **Liste** der dritten Potenzen `[1, 8, 27, 64, 125]`. (2 P)

b) Schreiben Sie eine Funktion `vektoraddition(v, w)`, die mit `map` zwei gleichlange Listen elementweise addiert und das Ergebnis als Liste zurückgibt. Beispiel:
`vektoraddition([1,2],[10,20]) → [11, 22]` (3 P)

c) Gegeben ist eine Liste von Wörtern `woerter`. Erzeugen Sie mit `filter` die Liste aller Wörter mit **mehr als vier Buchstaben**. (3 P)

d) Berechnen Sie mit `reduce` den **Durchschnitt** einer nichtleeren Zahlenliste `zahlen`. (3 P)

Aufgabe 4: Programmieren mit Rekursion (12 Punkte)

Verwenden Sie in dieser Aufgabe **keine Schleifen**, sondern ausschließlich Rekursion.

a) Schreiben Sie eine rekursive Funktion `umkehren(text)`, die eine Zeichenkette rückwärts zurückgibt. Beispiel: `umkehren("hallo") → "ollah"` (4 P)

b) Schreiben Sie eine rekursive Funktion `verschachtelte_summe(liste)`, die alle Zahlen in einer **beliebig tief verschachtelten** Liste aufsummiert. Beispiel: `verschachtelte_summe([1, 2, [3, 4], [5, [6]], 7]) → 28`. Hinweis: Mit `isinstance(x, list)` prüfen Sie, ob `x` eine Liste ist. (5 P)

c) Schreiben Sie eine rekursive Funktion höherer Ordnung `wiederhole(f, x, abbruch)`, die die Funktion `f` so lange wiederholt auf `x` anwendet, bis `abbruch(x)` den Wert `True` liefert, und dann `x` zurückgibt. (3 P)

Aufgabe 5: Funktional umschreiben & Currying (8 Punkte)

a) Gegeben ist folgender imperativer Code mit Schleife und Mutation:

```
namen = ["anna", "ben", "christina", "dora"]
```

```
doku = ""
for name in namen:
    if len(name) > 3:
        doku = doku + name.upper() + "\n"
print(doku)
```

Schreiben Sie den Code so um, dass doku **ohne Schleife** und **ohne Mutation** in einem einzigen Ausdruck aus map, filter und reduce entsteht. Der print-Befehl bleibt als einziger Nebeneffekt erhalten. (5 P)

b) Gegeben ist:

```
def potenz(basis, exponent):
    return basis ** exponent
```

Erzeugen Sie durch **Binden** eines Parameters (wie beim Currying aus der Vorlesung) mit einem lambda-Ausdruck eine neue Funktion quadrat, sodass `quadrat(5) → 25` gilt. Geben Sie außerdem an, was allgemein für die gecurryte Form `f_curry` einer zweistelligen Funktion `f` gilt: `f_curry(a1)(a2) = ?` (3 P)

Aufgabe Bonus: Decorators (+4 Punkte)

Schreiben Sie einen Decorator `rufezeichen`, der jede Funktion, die einen String zurückgibt, so verändert, dass an den Rückgabewert drei Ausrufezeichen ("!!!") angehängt werden. Dekorieren Sie damit anschließend per @-Syntax eine Funktion `begrueessung(name)`, die "Hallo, <name>" zurückgibt.