University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatik I
Einführung in die Programmierung

Zwischenprüfung HS18

Allgemeine Hinweise:

- Die maximale Punktzahl beträgt **90 Punkte**, erreichbar durch das Lösen aller Aufgaben.
- Die Bearbeitungszeit beträgt **90 Minuten**.
- Bitte überprüfen Sie, dass Sie alle 11 Seiten dieser Klausur erhalten haben.
- Benutzen Sie einen **schwarzen oder blauen, dokumentenechten Stift** für die Prüfung. Stifte mit **grüner oder roter** Farbe sowie **Bleistifte** sind nicht **nicht gestattet**. Betroffene Antworten werden bei der Bewertung nicht berücksichtigt.
- Lassen Sie die Blätter dieser Klausur zusammengeheftet.
- Bitte schreiben Sie Ihren **Nachnamen** und Ihre **Matrikelnummer** auf die dafür vorgesehene Markierung am Ende **jeder Seite**.
- Sie dürfen eine **handgeschriebene Formelsammlung** verwenden (DIN-A5, beidseitig beschrieben), die klar mit Ihrem Namen gekennzeichnet ist.
- Studierende, deren Muttersprache nicht Deutsch ist, dürfen ein **Wörterbuch** verwenden.
- Die Verwendung von zusätzlichen Materialien **ist nicht gestattet**. Sollten Sie zu unfairen Mitteln greifen, nicht genehmigte Ressourcen verwenden, oder von einem Kommilitonen abschreiben, wird die Klausur eingezogen und als nicht bestanden gewertet. Zusätzlich werden disziplinarische Massnahmen eingeleitet.
- Schreiben Sie Quelltexte in Python. Sie können dabei frei zwischen Version 2 und 3 und den entsprechenden Funktionen wählen. Es ist nicht erlaubt vordefinierte Funktionen zu verwenden, wenn deren Implementierung in der Aufgabenstellung gefordert ist.
- Sie finden am Ende der Prüfung eine Liste von hilfreichen Pythonfunktionen.
- Ändern Sie keine vorgegebene Methodensignaturen oder Variablennamen der Prüfung.

Bitte füllen Sie die folgenden Felder in **grossen Druckbuchstaben** aus und schreiben Sie **deutlich**:

Vorname: _____ Nachname: _____

Matrikelnummer: _____

Ich bestätige mit meiner Unterschrift:

- Ich habe die Hinweise gelesen und verstanden.
- Ich fühle mich körperlich und psychisch in der Lage an der Klausur teilzunehmen.

Unterschrift: _____ Datum: _____

Aufgabe 1	Aufgabe 2	Aufgabe 3	Aufgabe 4	Summe
20	30	20	20	90

Aufgabe 1: Allgemeine Fragen — 20 Punkte

Diese Aufgabe enthält einige kleine Pythonsnippets von denen jedes in der letzten Zeile einen Ausdruck enthält. Schreiben Sie den *Typ* und den *Wert* dieses Ausdrucks in die dafür vorgesehenen Felder. Lassen Sie das *Wert* Feld leer, wenn der Ausdruck keinen Wert hat. Wählen Sie im Fehlerfall den *NoneType* als Typ und schreiben "error" als Wert. Wählen Sie im Falle einer Endlosschleife den *NoneType* als Typ und schreiben "endless loop" als Wert.

Hinweis: Sie müssen keine Module nennen, geben Sie aber den genauen Typnamen an, z.B. 'int'.

Hinweis: Die Snippets werden getrennt ausgeführt und haben keinerlei Seiteneffekte aufeinander.

Hinweis: Lesen Sie die Snippets sehr aufmerksam. Die Antwort ist nicht immer offensichtlich.

a) 2 Punkte

```
1.2 + 3 + "5"
```

Typ: Wert:

b) 2 Punkte

```
i = [1, 2, 3]
i + [4,5]
i.append([6, 7])
i.extend([8, 9])
i
```

Typ: Wert:

c) 2 Punkte

```
b1 = "4321"
b2 = "12343"
b1[3:4] + b2
```

Typ: Wert:

d) 2 Punkte

```
False or False and True or False
```

Typ: Wert:

e) 2 Punkte

```
tmp = (lambda x: x//2, 13)
tmp[0](tmp[1])
```

Typ: Wert:

f) 2 Punkte

```
def e():
   for i in range(0, 10, 2):
     if i == 10:
       return i
e()
```

Typ: Wert:

g) 2 Punkte

```
name = "John doe"
name[5] = "D"
name
```

Typ: Wert:

h) 2 Punkte

```
h = {'a': 1, "b": 2.3}
tmp = None
for x in h:
   tmp = x
   break
tmp
```

Typ: Wert:

i) 2 Punkte

```
i = 0
while i <= 100:
    i+=1
i
```

Typ: Wert:

j) 2 Punkte

```
j = 0
def fun(n):
    j = n
fun(3)
j
```

Typ: Wert:

Aufgabe 2: Kontrollfluss und Datenstrukturen 30 Punkte

In dieser Aufgabe werden Sie einige einfache Hilfsfunktionen schreiben. Jede Unteraufgabe stellt ein kleines Programmierproblem vor. Die Beispiele illustrieren das erwartete Verhalten das Ihre Implementierung erfüllen muss.

Hinweis: Sie müssen Funktionsargumente nicht auf `None` überprüfen, jedoch müssen Sie Sonderfälle des Argumenttyps behandeln (z.B.: negative Werte oder leere Strings).

a) Kontrollfluss 5 Punkte

Schreiben Sie eine Funktion, die ein numerisches Alter in eine Beschreibung überführt. Unter drei Jahren werden *Kinder* als *Kleinkinder* bezeichnet. Ab 18 Jahren gilt ein Mensch als *Erwachsener*, ab 65 Jahren als *Senior*. Sie können davon ausgehen, dass `age` immer grösser als null ist.

```
def get_age_desc(age):
```

```
assert get_age_desc(1) == "Kleinkind"
assert get_age_desc(3) == "Kind"
assert get_age_desc(14) == "Kind"
assert get_age_desc(18) == "Erwachsener"
assert get_age_desc(65) == "Senior"
```

b) Iteration 5 Punkte

Die Funktion `compute_max_diff` erhält eine Liste mit Zahlen als Argument. Implementieren Sie die Funktion und berechnen Sie die höchste absolute Differenz zwischen zwei aufeinanderfolgenden Zahlen in der Liste. Sie dürfen in Ihrer Lösung die vordefinierte Funktion `abs` verwenden.

```
def compute_max_diff(numbers):
```

```
assert compute_max_diff([]) == 0
assert compute_max_diff([1]) == 0
assert compute_max_diff([2, -1]) == 3
assert compute_max_diff([2, 7, 5, 14, 12, 14]) == 9
```

c) Gerade/Ungerade Summe 5 Punkte

Implementieren Sie eine Funktion, welche alle Zahlen in einer Liste von Ganzzahlen aggregiert. Die geraden Zahlen sollen hierbei *addiert*, die ungeraden Zahlen *subtrahiert* werden. Geben Sie ein Tuple zurück, das die Länge der Liste und den berechneten Wert enthält.

```
def sum_even_and_odd(numbers):
```

```
assert sum_even_and_odd([]) == (0, 0)
assert sum_even_and_odd([2,4,7]) == (3, -1) # 3 values, 2+4-7=-1
```

d) Funktionale Programmierung 5 Punkte

Implementieren Sie eine Funktion, die eine Operation `op` auf jedes Element einer Liste `l` anwendet. Geben Sie eine neue Liste zurück und verändern Sie nicht die ursprüngliche Liste.

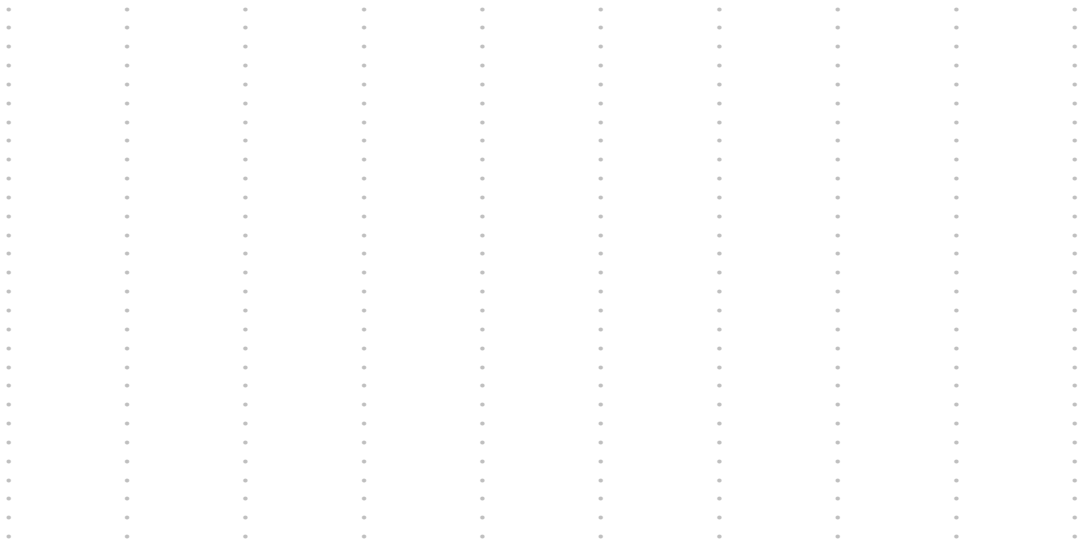
```
def app(l, op):
```

```
l = [1, 2, 3]
def double(x): return x * 2
assert app([], double) == []
assert app(l, double) == [2, 4, 6]
assert app(l, lambda x: x * 3) == [3, 6, 9]
assert l == [1, 2, 3]
```

e) Zählung 5 Punkte

Schreiben Sie eine Funktion, welche die Häufigkeit aller enthaltener Zeichen eines Strings zählt. Geben Sie das Ergebnis als Dictionary zurück.

```
def count_chars(s):
```



```
assert count_chars("") == {}
assert count_chars("aA .") == {"a": 1, "A": 1, " ": 1, ".": 1}
assert count_chars("abbCaabb") == {"a": 3, "b": 4, "C": 0}
```

f) Funktionsaufrufe 5 Punkte

Schreiben Sie eine Funktion, welche die Häufigkeit eines beliebigen Zeichens in einem String zählt. Verwenden Sie die Funktion `count_chars` aus der vorigen Aufgabe.

Hinweis: Sie können davon ausgehen, dass `c` immer genau ein Zeichen enthält.

```
def count_occurrences(s, c):
```

```
assert count_occurrences("", "a") == 0
assert count_occurrences("x", "b") == 0
assert count_occurrences("abbccc", "c") == 3
```

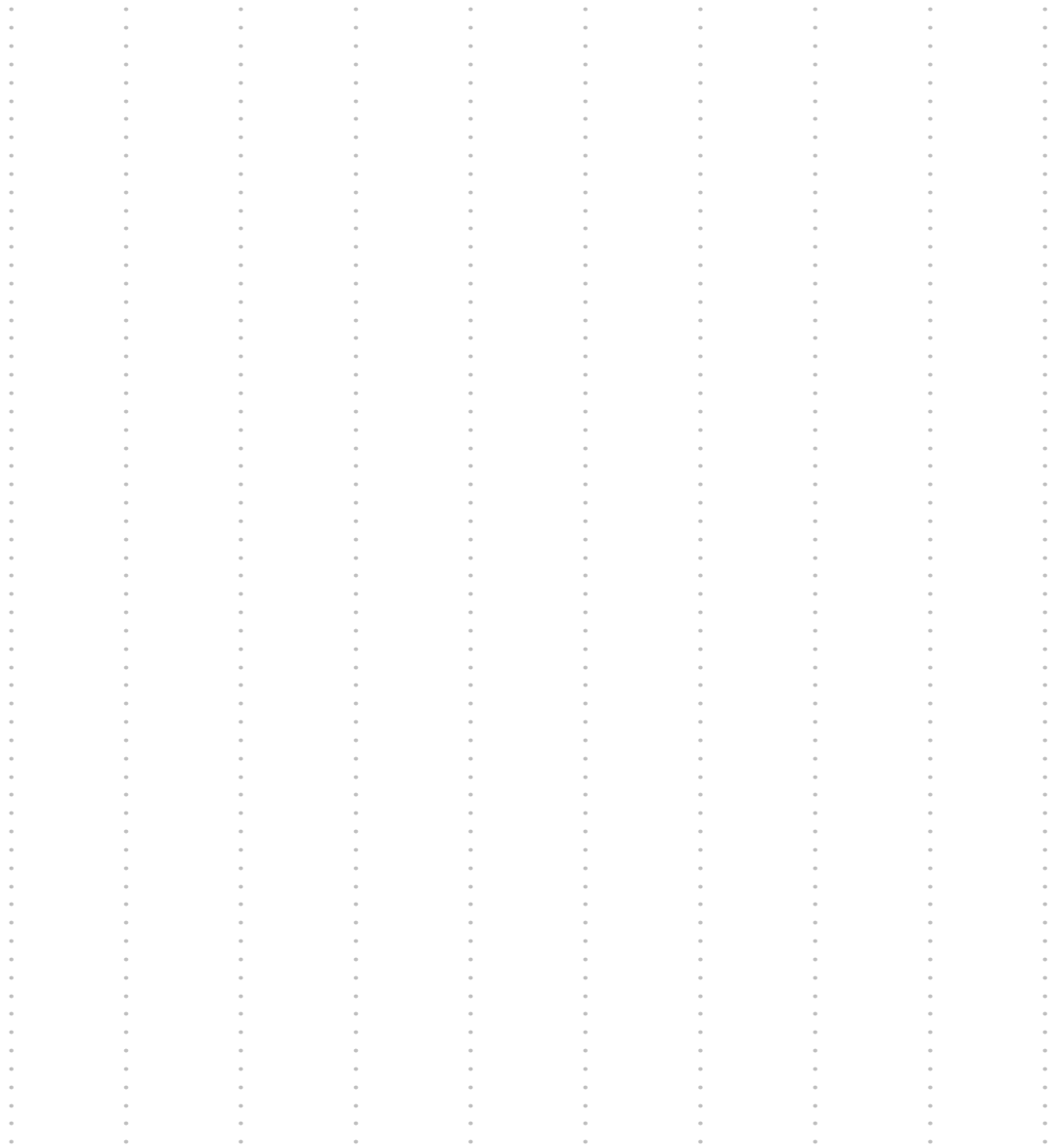
Aufgabe 3: Rekursive Funktionen 20 Punkte

Implementieren Sie die Funktion `nested_sum`, die alle Elemente einer Liste `l` summiert. Die Elemente sind Ganzzahlen oder verschachtelte Listen, die dem gleichen Muster folgen. Die berechnete Summe soll die Verschachtelungstiefe als Gewicht der Elemente verwenden, beispielsweise sollen alle Elemente auf der ersten Ebene das Gewicht 1 haben, auf der zweiten Ebene Gewicht 2, und so weiter. Sie finden konkrete Beispiele von solchen Listen und den Berechnungen am Ende des Blocks.

Hinweis: Nehmen Sie an, dass `l` immer eine valide Liste ist, d.h., nur Ganzzahlen und andere verschachtelte Listen enthält. `l` kann auch leer sein. Verletzen Sie diese Annahmen nicht selbst.

Hinweis: Ihre Lösung muss rekursiv sein.

```
def nested_sum(l, depth=1):
```



```
assert nested_sum([]) == 0
assert nested_sum([1, 2, 3]) == 6 # 1*(1+2+3)
assert nested_sum([1, [2]]) == 5 # 1*(1+2*(2))
assert nested_sum([1, [2, [3]]]) == 23 # 1*(1+2*(2+3*(3)))
assert nested_sum([[1, 2], 3, [4, 5]]) == 27 # 1*(2*(1+2)+3+2*(4+5))
```

Aufgabe 4: Dateien — 20 Punkte

Sie arbeiten in der Universitätsverwaltung und möchten die Klausurergebnisse aller aktuellen Studenten analysieren. Der Administrator hat die Klausurdaten in eine Datei exportiert, Sie müssen die Daten jedoch noch vorbereiten, um sie einfacher in Ihrem Programm benutzen zu können.

Ihre Datei `results.csv` hat das folgende Format. Jede Zeile enthält ein Klausurergebnis und besteht aus drei Feldern, die mit Komma getrennt sind: Name, Kurs und erreichte Note.

```
Hans,Info1,5.5
Hans,Economics,5
Petra,Info1,5.25
Petra,Economics,4.75
Petra,Math,6
Martin,Info1,5.75
```

Implementieren die Funktion `read` welche die Inhalte von solchen Dateien nutzbar macht. Die Funktion soll die Datei lesen, welche durch `path` referenziert wird. Trennen Sie jede Zeile in die drei Felder auf und speichern diese in einem Dictionary. Das Dictionary soll den *Studentennamen* als Key verwenden und die Liste aller *Ergebnisse* als Wert; diese Liste soll den *Kursnamen* und die erreichte *Note* als *Tuple* enthalten (siehe `assert` Beispiele).

Hinweis: Sie können davon ausgehen, dass `path` immer auf eine existierende Datei zeigt, die strikt dem Schema entspricht. Die Note ist immer ein valider `float`, verwenden Sie sie entsprechend.

```
def read(path):
```

```
assert read("results.csv") == {"Hans": [("Info1", 5.5),
  ("Economics", 5)], "Petra": [("Info1", 5.25), ("Economics", 4.75),
  ("Math", 6)], "Martin": ["Info1", 5.75]}
```

Nützliche Pythonfunktionen

Strings

str.isupper() / str.islower() Gibt `True` zurück, falls alle Zeichen des nicht leeren Strings `str` Grossbuchstaben/Kleinbuchstaben sind, andernfalls `False`.

str.split(sep) Bricht einen Strings `str` bei jedem Vorkommen von `sep` in einzelne Wörter. `sep` ist optional, standardmässig werden die Wörter durch Whitespacezeichen getrennt (space, tab, newline, return, formfeed).

str.join(words) Erstellt einen String aus den Wörtern in `words` durch Aneinanderreihung. Die Wörter werden mit dem Wert von `str` verbunden.

str.isalpha() / str.isdigit() Ist `True`, wenn alle Zeichen eines nicht leeren Strings Buchstaben/Zahlen sind, sonst `False`.

str.startswith(prefix) Ist `True`, wenn der String `str` mit `prefix` beginnt, sonst `False`.

str.endswith(suffix) Ist `True`, wenn der String `str` mit `suffix` endet, sonst `False`.

string.find(x) Ermittelt den Startindex von `x`, wenn es im String vorkommt, sonst `-1`.

Lists

list.append(x) Hängt ein Element `x` an das Ende der Liste `a` an; äquivalent zu `a[len(a):] = [x]`.

list.remove(x) Entfernt das erste Element der Liste, dessen Wert `x` ist. Wirft einen Fehler, falls kein solches Element vorhanden ist.

list.index(x) Gibt den Index des ersten Elements zurück, dessen Wert `x` ist. Wirft einen Fehler, falls kein solches Element vorhanden ist.

list.count(x) Zählt wie häufig `x` in einer Liste vorkommt.

Dictionaries

dict.has_key(key) `True`, wenn `key` im Dictionary vorhanden ist, sonst `False`.

dict.keys() Gibt eine Liste aller Keys zurück, die im Dictionary `dict` definiert sind.

dict.items() Gibt eine Liste aller `(Key, Value)` Tuples des Dictionary zurück.

dict.values() Gibt eine Liste aller Dictionary Values zurück.

dict.get(key, default=None) Gibt es den Wert zurück, der mit `key` assoziiert ist oder `default`, wenn der Key nicht existiert.

dict.pop(key) Entfernt `key` aus dem Dictionary und gibt dessen vorherigen Wert zurück.

Files

open(filename, 'r') Öffnet die Datei `filename` zum Lesen und gibt ein Dateiobjekt zurück.

open(filename, 'w') Öffnet die Datei `filename` zum Schreiben und gibt ein Dateiobjekt zurück.

f.close() Schliesst das Dateiobjekt `f`.

f.readline() Gibt die nächste Zeile des Dateiobjekts `f` zurück.

f.readlines() Gibt alle Zeilen des Dateiobjekts `f` zurück.

os.path.isfile(file) Ist `True`, wenn `file` existiert und eine reguläre Datei ist.

Other

isinstance(obj, type) Ist `True`, wenn der Typ von `obj` kompatibel zu `type` ist, ansonsten `False`.

len(obj) Gibt die Länge eines Objekts zurück. `obj` kann eine Sequenz sein (z.B.: string, list, etc.) oder eine collection (z.B.: dictionary).

sorted(sequence) Erzeugt aus den Elementen der Sequenz eine neue sortierte Liste.

abs(x) Erzeugt aus den Elementen der Sequenz eine neue sortierte Liste.