

Prüfung vom 5. Februar 2021

90 Minuten / 82 Punkte

Name, Vorname: _____

Allgemeine Hinweise:

- 1) Erlaubte Hilfsmittel: 2 Seiten A4 mit Notizen
- 2) Nicht erlaubt: Elektronische Geräte (Handy, Taschenrechner, Kopfhörer), Kommunikation mit anderen Personen
- 3) Die Antworten sollten direkt auf den Aufgabenblättern geschrieben werden.
- 4) Auf alle allfälligen Zusatzblätter bitte mit den Namen schreiben.
- 5) Bitte nicht mit roter Farbe schreiben.

Viel Erfolg!

Alle Aufgaben sollen ausschliesslich mit den Elementen der Sprache Python, insbesondere ohne die Verwendung von Packages wie pandas, numpy, scipy, sympy, etc. gelöst werden.

Die senkrechten Linien bei den Programmieraufgaben können zur Orientierung bei der Einrückung dienen.

Der für die Programme zur Verfügung gestellte Platz stellt *keinen* Hinweis auf die Länge der erwarteten Lösung dar, sondern orientiert sich vor allem an der Seitenaufteilung der Aufgaben.

Der Platz auf Seite 2 kann bei Bedarf genutzt werden, wenn der Platz bei einer Aufgabe nicht ausreicht. Bringe bitte dann bei der entsprechenden Aufgabe den Hinweis «s. Seite 2» an.

Du brauchst eigenen Funktionen *keine Doc Strings* zu notieren.

Die einzelnen Teilaufgaben können auch gelöst werden, wenn vorherige Teilaufgaben nicht oder unvollständig gelöst wurden. Funktionen dürfen aufgerufen werden, auch wenn sie nicht (vollständig) programmiert sind.

--	--	--	--	--	--	--

Aufgabe 1: Programm-Analyse**(13 Punkte)**

Was geben die folgenden Python-Programme aus? Notiere die Ausgaben jeweils auf den Linien neben den print-Anweisungen.

a) [2 Punkte]

```
s = 0
sq = s
for i in range(3):
    s, sq = s + i, sq + i * i
print(s, sq)
```

b) [3 Punkte]

```
def fun(a, b, c):
    a = b + c
    b = 0
    return a - c
```

```
a, b, c = 1, 2, 3
c = fun(a, b, c)
print(a, b, c)
```

c) [6 Punkte]

```
lst = [[1, 2, 3], [4, 5], [6]]
lst[2].append(len(lst[-1]))
print(lst)
```

```
l2 = lst[1]
l2[0] = 0
del lst[0]
print(l2)
```

```
print(lst)
```

d) [2 Punkte]

```
x = 5 ** 2 + 1
print(x < 20 or x > 30)
```

```
print((x < 20) == (x > 30))
```

Aufgabe 2: Comprehensions**(4 Punkte)**

Schreibe zu den folgenden beiden Python-Programmen gleichwertige Einzeiler mittels *Comprehensions* auf.

a) [2 Punkte]

```
lst = []
for x in range(n):
    if x % 3 == 0:
        lst.append(x)
```

```
lst =
```

b) [2 Punkte]

```
lst = []
for x in range(n):
    lst.append(x ** 3)
```

```
lst =
```

Aufgabe 3: Nächste Primzahl

(14 Punkte)

In einer der Übungsaufgaben habt Ihr eine Funktion `is_prime(n)` programmiert, die für eine als Parameter gegebene Zahl `n` entscheidet, ob es sich um eine Primzahl handelt. Diese Funktion setzen wir nun als *aufzurufbar* voraus.

- a) [4 Punkte] Programmiere die Python-Funktion `next_prime(n)`, welche die erste auf die Zahl `n` folgende Primzahl ermittelt:

```
>>> next_prime(8)
11
>>> next_prime(11)
13
```

```
def next_prime(n):
```

- b) [10 Punkte] Programmiere eine Python-Funktion `prime_dist(start, dist)`, die aus dem Zahlenintervall `[start ... 10·start]` das erste Paar aufeinanderfolgender Primzahlen mit einem Abstand von mindestens `dist` herausucht. Gibt es kein solches Paar, soll das Ergebnis `None` sein:
- Erste Primzahl ist `start` selbst, oder – wenn `start` nicht prim ist – die erste darauffolgende Primzahl.
 - Mittels `next_prime` wird nun als zweite Primzahl die darauffolgende Primzahl gesucht.
 - Solange der Abstand dieser beiden Primzahlen kleiner als `dist` und die grössere der Primzahlen kleiner als das 10-fache des Wertes von `start` ist, wird wieder die nächste Primzahl gesucht und so weiter.
 - Ist der Abstand zweier so gefundener aufeinanderfolgender Primzahlen mindestens gleich `dist`, ist das Ergebnis der Funktion ein Tupel aus gerade diesen beiden Primzahlen.

```
>>> prime_dist(3, 6)
(23, 29)
>>> prime_dist(1000, 10)
(1021, 1031)
>>> prime_dist(1000, 20)
(1129, 1151)
>>> prime_dist(1000, 35)
(9551, 9587)
>>> prime_dist(1000, 37)
>>>
>>> prime_dist(1000000, 100)
(1098847, 1098953)
```

```
def prime_dist(start, dist):
```

(16 Punkte)

- Eine tote Zelle mit genau zwei lebenden Nachbarn wird in der Folgegeneration neu geboren (und lebt dann)
- Lebende Zellen mit weniger als zwei lebenden Nachbarn sterben in der Folgegeneration an Einsamkeit
- Eine lebende Zelle mit zwei oder drei lebenden Nachbarn bleibt in der Folgegeneration am Leben
- Lebende Zellen mit mehr als drei lebenden Nachbarn sterben in der Folgegeneration an Überbevölkerung

		x		x	x	x		
					x	x	x	
						x	x	x
							x	x

```
[ '.....', '###.##...', '.....##...', '.....##.', '.....##.', '.....' ]
```

- ```
>>> living_neighbours(grid, 1, 1)
1
>>> living_neighbours(grid, 1, 2)
2
>>> living_neighbours(grid, 3, 1)
0
```

```
def living_neighbours(grid, row, col):
```

---

---

---

---

---

---

- b) [10 Punkte] Programmiere eine Python-Funktion `step(grid)`, die für ein Grid die nächste Generation gemäss den angegebenen Regeln berechnet. Die Funktion `living_neighbours` darf dabei aufgerufen werden.

```
def step(grid):
```

```
.....
```

### Aufgabe 5: IoT Data Check

**(16 Punkte)**

Ein Temperatursensor meldet regelmässig Messdaten, die in einer Datei gesammelt werden. Diese Datei enthält durch Komma getrennt *Datum* und *Uhrzeit* der Messung, die gemessene *Temperatur* in °Celsius und einen *Status*, der angibt, ob die Messung korrekt ist («ok»), oder ob der Sensor ausgefallen ist («fault») oder ausgeschaltet («off»).

Ein Beispiel für eine solche Datei:

```
2021-01-23, 14:05:12, 21, ok
2021-01-23, 14:10:54, 19, ok
2021-01-23, 14:15:23, 0, off
2021-01-23, 14:20:05, 19, ok
2021-01-23, 14:25:23, 128, fault
2021-01-23, 14:30:41, 20, ok
2021-01-23, 14:35:51, 46, ok
2021-01-23, 14:40:22, 20, off
2021-01-23, 14:45:34, -11, ok
```

Eine Zeile bezeichnen wir als *korrekt*, wenn sie folgende Bedingungen erfüllt:

- alle Temperaturen mit *Status* «ok» sind mindestens -10°C und höchstens 45°C
- alle Temperaturen mit *Status* «off» haben den Wert 0°C
- Temperaturen mit *Status* «fault» dürfen beliebig sein

In der oben gezeigten Beispieldatei sind die ersten 6 Zeilen *korrekt* und die letzten drei nicht.

a) [5 Punkte] Programmieren Sie eine Python-Funktion `is_ok(temperature, status)`, die einen Wahrheitswert zurückgibt, der zeigt, ob die Kombination aus `temperature` und `status` gemäss diesen Regeln *korrekt* ist:

```
>>> is_ok(30, 'ok')
True
>>> is_ok(-10, 'ok')
True
>>> is_ok(-11, 'ok')
False
>>> is_ok(-11, 'off')
False
>>> is_ok(0, 'off')
True
>>> is_ok(90, 'fault')
True
```

```
def is_ok(temperature, status):
```

[illegible]



- b) [8 Punkte] Programmieren Sie eine Python-Funktion `count_errors(file_name)`, die eine Datei in der beschriebenen Form liest und die *Anzahl nicht korrekter Zeilen* retourniert. Für die als Beispiel gezeigte Datei wäre das Ergebnis 3. Die Funktion soll die Funktion `is_ok` aus Aufgabe a) geeignet aufrufen.

```
>>> count_errors('iot.txt')
3
```

```
def count_errors(file_name):
```

- c) [3 Punkte] Programmiere eine Python-Funktion `count_errors_mult(file_names)`, die für eine Liste von Datei-Namen die Gesamtzahl der *nicht korrekten* Zeilen in allen diesen Dateien ermittelt. Funktionen aus den beiden Teilaufgaben a) und b) dürfen aufgerufen werden.

```
>>> count_errors_mult(['iot.txt', 'other_iot.txt', 'my_dat.txt'])
9
```

```
def count_errors_mult(file_names):
```

### Aufgabe 6: Einkaufsstatistik

(19 Punkte)

Ein Versandhändler führt eine Datei mit Informationen über abgeschlossene und erledigte Verkäufe. Diese Datei enthält eine Kopfzeile und dann sehr viele Zeilen, die jeweils einen Artikelverkauf repräsentieren. Hier ein (kleines) Beispiel:

| Auftrag | Datum     | Kunde | Artikel | Stk-Preis | Anzahl | Bezahldatum | Lieferdatum |
|---------|-----------|-------|---------|-----------|--------|-------------|-------------|
| 0815    | 23.4.2020 | 12345 | 50000   | 32.45     | 1      | 23.4.2020   | 27.4.2020   |
| 0815    | 23.4.2020 | 12345 | 123     | 160.00    | 2      | 23.4.2020   | 27.4.2020   |
| 4711    | 23.4.2020 | 3333  | 13246   | 65.15     | 2      | 25.4.2020   | 27.4.2020   |
| 1234    | 23.4.2020 | 12345 | 3111    | 45.00     | 1      | 23.4.2020   | 25.4.2020   |
| 6413    | 28.5.2020 | 12345 | 50000   | 32.45     | 2      | 30.5.2020   | 1.6.2020    |

Die einzelnen Spalten sind in jeder Zeile jeweils durch Tabulator-Zeichen ('\t') getrennt.

Dein Auftrag ist es, eine Python-Funktion `articles(file_name, customer)` zu programmieren, die aus einer solchen Datei alle je vom Kunden `customer` gekauften Artikel sucht und die gekauften Stückzahlen summiert:

```
>>> articles('sales.txt', 12345)
Artikel -> Anzahl
50000 -> 3
123 -> 2
3111 -> 1
```

```
>>> articles('sales.txt', 3333)
Artikel -> Anzahl
13246 -> 2
```

```
>>> articles('sales.txt', 4)
keine Bestellungen
```

Die Funktion `articles` muss dazu die gesuchten Daten (relevante Artikel und Anzahl Verkäufe an den zu untersuchenden Kunden) zunächst in einer Datenstruktur sammeln. Anschliessend kann aus dieser Datenstruktur die Ausgabe erzeugt werden.

Die Datei ist so gross, dass sie nur Zeile für Zeile gelesen und analysiert werden darf, weil sonst zu viel Speicher belegt würde.

Die Funktion `articles(file_name, customer)` kann also folgendermassen aussehen:

```
def articles(file_name, customer_no):
 data = collect_sales_data(file_name, customer_no)
 print_sales_data(data)
```

- a) [3 Punkte] Beschreibe die Datenstruktur, die durch `collect_sales_data()` aufgebaut werden soll. Welche Python-Datenstruktur(en) würdest Du verwenden? Warum gerade diese?

- b) [12 Punkte] Programmiere die Funktion `collect_sales_data(file_name, customer_no)`. Die Funktion soll zunächst die Kopfzeile der Datei einlesen und analysieren, um herauszufinden an welchen Stellen der folgenden Zeilen («in welchen Spalten») sich Kundennummer («Kunde»), Artikelnummer («Artikel») und Bestellmenge («Anzahl») befinden. Anschliessend müssen diese drei Werte aus den restlichen Zeilen der Datei herausgelesen werden.

```
def collect_sales_data(file_name, customer_no):
```

```
.....
```

c) [4 Punkte] Programmiere in Python die Funktion `print_sales_data(data)`.  
`def print_sales_data(data):`

Vertical dotted lines for code entry.