Aufgabenblatt 6

Allgemeine Informationen zum Aufgabenblatt:

- Die Programme müssen syntaktisch korrekt sein. Achten Sie außerdem darauf, dass die Beispiele aus der Angabe zu keinen Abstürzen führen.
- Wenn Fehler auftreten ziehen wir abhängig von der Schwere der Fehler Punkte ab. Bei entsprechend kleinen Fehlern können auch gar keine Punkte abgezogen werden. Geben Sie daher die bestmögliche eigene Lösung ab, auch wenn diese nicht vollständig richtig ist.
- Ihre Programme sollen demonstrieren, dass Sie die Themen des Aufgabenblattes beherrschen. Verwenden sie daher keine "Abkürzungen" und beschränken Sie sich auf die Konstrukte die in der Vorlesung vorgestellt wurden.
- Um das Testen einzelner Aufgaben zu erleichtern sollen alle Aufgaben als entsprechende Funktionen implementiert werden. Bitte achten Sie darauf, dass Sie in der Abgabedatei keinen Code außerhalb der in den jeweiligen Aufgaben definierten Funktionen erstellen. Sie finden entsprechende Kommentare in der Angabedatei.
- Bei den meisten Angaben sind explizite Annahmen definiert. Sie können davon ausgehen, dass diese Annahmen stets eingehalten werden. D.h. Sie müssen deren Gültigkeit weder überprüfen, noch muss Ihre Implementierung für Werte welche diese Annahmen nicht erfüllen funktionieren.
- Type hints: Zur besseren Beschreibung und Dokumentation der verwendeten Funktionen verwenden wir ab diesem Aufgabenblatt sogenannte Type-Hints in den Funktionsköpfen. Diese beschreiben die intendierten Typen der Parameter und Rückgabewerte. Die von uns verwendeten Type Hints stehen erst ab **Python 3.10** zur Verfügung.
- Das Aufgabenblatt enthält 4 Aufgaben, auf welche Sie insgesamt 5 Punkte erhalten können.

In diesem Aufgabenblatt werden folgende Themen behandelt:

- Dictionaries
- Lesen und Schreiben von Dateien
- Klassen (Objektorientierung)
- O-Notation

Aufgabe 1 (Dictionaries) [1 Punkt]

Ziel dieser Aufgabe ist es, den Umgang mit verschachtelten Dictionaries zu üben. In der Python Datei `blatt6.py` zur Angabe ist dafür in der Funktion `things_in_space` ein Dictionary `space_probes` vorgegeben, das Informationen über eine kleine Auswahl an Raumsonden enthält[1]. Dabei bildet der Name der Sonde den Schlüssel, und jeder Eintrag ist wiederum ein Dictionary. Dieses verschachtelte Dictionary enthält immer die Schlüssel `'mission'`, `'launch_mass'`, `'launch_site'`, `'launch_rocket'` und `'operator'`, und kann zusätzlich die Schlüssel `'visited'` und `'destination'` enthalten. Dabei enthält `'mission'` nochmals ein Dictionary, welches immer den Schlüssel `'launch'` enthält, und zusätzlich einen der beiden Schlüssel `'end'` und `'failed'` enthalten kann.
Sie dürfen zur Lösung der Aufgaben alle in der Vorlesung behandelten Sprachkonstrukte verwenden, sowie die in Python verfügbaren Funktionen für Kollektionen. Erstellen Sie die geforderte Lösung dynamisch, und geben Sie nicht einfach die fixen Werte für die konkrete Angabe an. Achten Sie also darauf, dass Ihre Lösung auch für andere Dictionaries welche die oben beschriebene Struktur besitzen funktioniert. Achten Sie bei Ihren Lösungen auch auf die oben beschriebenen Annahmen, welche Schlüssel immer verfügbar sind, und welche fehlen können.
Ihre Lösung darf das Dictionary `space_probes` und dessen Inhalt nicht verändern. Achten Sie darauf, dies auch bei den verschachtelten Datenstrukturen zu befolgen.
Implementieren Sie Ihre Lösung innerhalb der Funktion `things_in_space()`.

- Geben Sie das Startgewicht (`'launch_mass'`) der schwersten Sonde aus.

  Erwartete Ausgabe:
  `8250`

- Erstellen Sie eine Menge (`set`) aller Ziele (`'destination'`) von Sonden. Speichern Sie eine Referenz auf die Menge in der Variable `destinations`, und geben Sie die Menge mittels `print` aus.

  Mögliche Ausgabe (Reihenfolge in Mengen kann variieren):
  `{'Wild 2', 'Venus', 'Saturn', 'Komet 67P', 'Mond', 'Mars', 'Jupiter'}`

- Raumsonden deren Mission bereits erfolgreich abgeschlossen wurde sind daran erkennbar, dass das Dictionary `'mission'` den Schlüssel `'end'` enthält. Geben Sie für all diese Raumsonden eine Zeile der Form
  `Name: Jahre`
  aus, wobei `Name` der Name der Sonde ist und `Jahre` die Differenz der Werte zu `'end'` und `'launch'` in `'mission'`. Sortieren Sie die Zeilen aufsteigend nach dem Namen der Sonde.

  Erwartete Ausgabe:
  ```
  Akatsuki: 14
  Cassini: 20
  Chang'e 5: 1
  Galileo: 14
  Rosetta: 12
  Stardust: 12
  ```

[1] Die Informationen sind grob aus Wikipedia zusammengetragen, und teilweise aus Platzgründen am Angabezettel gekürzt. Wir bitten dadurch entstandene Ungenauigkeiten zu entschuldigen.

- Erstellen Sie ein Dictionary, welches einen Eintrag für jeden Planeten besitzt, der schon mindestens einmal Ziel einer Sonde war. Der Schlüssel jedes Eintrags ist der Name des Planeten, und der Wert des Eintrags ein verschachteltes Dictionary. Dieses verschachtelte Dictionary enthält nun einen Eintrag für jede Sonde, welche den jeweiligen Planeten besucht hat. Die Schlüssel der Einträge sind die Namen der Sonden, und die Werte sind erneut ein verschachteltes Dictionary, welches Informationen über die Sonde enthält. Jedes dieser "innersten" Dictionaries besitzt die drei Schlüssel `'start'`, `'rocket'` und `'agency'`. Die zugehörigen Werte sind das Jahr der Starts der Sonde (`'launch'` in `'mission'`), die Rakete welche die Sonde gestartet hat (`'launch_rocket'`) und der Betreiber der Sonde `'operator'`.

  Sie können zum Lösen der Aufgabe die zuvor berechnete Menge `destinations` sowie die in der Angabedatei vordefinierte Menge `planets` verwenden.

  Speichern Sie das Ergebnis in einer Variable `destination_details`, und geben Sie es mittels `print` aus.

  Erwartetes Ergebnis:
  Das vollständige Ergebnis finden Sie als Kommentar in der Datei `blatt6.py`, aus Platzgründen hier nur ein Eintrag des Dictionaries als Beispiel:

  ```
  { ...
  'Venus': {'Akatsuki': {'start': 2010, 'rocket': 'H-2', 'agency': 'JAXA'}}
    ... }
  ```

Aufgabe 2 (Lesen und Schreiben von Dateien) [2 Punkte]

Ziel dieser Aufgabe ist es, das Lesen und Schreiben von Dateien auszuprobieren. Wir erweitern dazu unser Programm der Kaffeemaschine (Aufgabenblatt 2) um die Fähigkeit, durchgeführte Aktionen (einschalten, Wasser nachfüllen, Bohnen nachfüllen, entkalken, Kaffee machen, ausschalten) zu protokollieren, sowie beim Ausschalten die aktuelle Wasser- und Bohnenmenge zu speichern.
Das Programm der Kaffeemaschine ist dazu in `blatt6.py` bereits vorgegeben. Im Unterschied zu Aufgabenblatt 2 wurde die Maschine diesmal mit Hilfe einer Klasse und Methoden implementiert, alle Abläufe sind aber unverändert geblieben. Die Maschine zählt nun allerdings die Anzahl der Brühvorgänge mit, und verwaltet für die Wartungstätigkeiten Bohnen/Wasser nachfüllen und entkalken auch wann diese zuletzt durchgeführt wurden. Außerdem wurden an den nötigen Stellen bereits Funktionsaufrufe zum Speichern und Laden der Informationen in/aus einer Datei implementiert. Von diesen Funktionen sind jedoch ausschließlich die Funktionsköpfe vorhanden.
Ihre Aufgabe ist es, diese Funktionen zu implementieren. Vor der genauen Aufgabenstellungen zu den einzelnen Funktionen folgt zuerst eine Zusammenfassung, welche Informationen in welchem Format in der Protokolldatei gespeichert werden.

Protokolldatei Die Kaffeemaschine verwendet zum Speichern der Informationen eine Datei `maintenance.log`. Diese enthält eine Auflistung aller an der Maschine durchgeführen Operationen. Zu jeder Operation gibt es eine Zeile im Format
`[Zeitstempel]: Aktion`
Der `Zeitstempel` ist immer eine Zeitangabe im Format `YYYY-MM-DD hh:mm:ss`. Abhängig von der protokollierten Tätigkeit ist `Aktion` folgender String:

Tätigkeit	String
Einschalten	`'switched on'`
Bohnen nachfüllen	`'beans refill (<Menge>g)'`
Wasser nachfüllen	`'water refill (<Menge>ml)'`
Kaffee machen	`'coffee made (<size>)[water: <WMenge>; beans: <BMenge>]'`
Entkalken	`'decalcify'`
Ausschalten	`'switched off (water: <wasser>;beans: <bohnen>;decalc: <k>)'`

Dabei ist `<Menge>` die Menge der nachgefüllten Bohnen/des nachgefüllten Wassers in der jeweiligen Einheit, `<size>` die Anzahl der durchgeführen Aufgüsse (1, 2 oder 3), und `<WMenge>` und `<BMenge>` die tatsächlich verwendete Menge an Wasser bzw. Bohnen. Weiters ist `<wasser>` und `<bohnen>` die Wasser- bzw. Bohnenmenge die beim Ausschalten noch in der Maschine verfügbar sind, und `<k>` die Anzahl der gemachten Kaffees seit dem letzten Entkalken.
Ein Beispiel für eine solche Datei ist:

```
[2025-04-16 12:59:12]: switched on
[2025-04-16 12:59:22]: coffee made (2)[water: 150; beans: 15]
[2025-04-16 12:59:27]: water refill (240ml)
[2025-04-16 12:59:29]: switched off (water: 290;beans: 15;decalc: 1)
[2025-04-16 13:00:11]: switched on
[2025-04-16 13:00:58]: beans refill (70g)
[2025-04-16 13:01:00]: switched off (water: 90;beans: 85;decalc: 1)
[2025-04-16 13:01:04]: switched on
[2025-04-16 13:02:59]: switched on
[2025-04-16 13:03:03]: switched off (water: 200;beans: 30;decalc: 0)
```

```
[2025-04-16 13:03:06]: switched on
[2025-04-16 13:03:33]: coffee made (2)[water: 150; beans: 30]
[2025-04-16 14:47:32]: beans refill (40g)
[2025-04-16 14:47:37]: decalcify
[2025-04-16 14:47:39]: switched off (water: 50;beans: 40;decalc: 1)
```

Hinweis: Dass die Maschine, wie im gezeigten Beispiel, zwei mal direkt hintereinander angeschalten wird, kann passieren, wenn das Programm nicht mittels 'quit' beendete wurde. Darum ändert sich auch die protokollierte Wasser- und Bohnenrestmenge in der Maschine, ohne dass etwas verbraucht oder nachgefüllt wurde. Wir prüfen die Datei aber nicht auf die Plausibilität der Abläufe.

Allgemeine Funktionalität Die Kaffeemaschine weist folgendes Verhalten auf:
Im laufenden Betrieb protokolliert die Maschine die zuvor beschriebenen Aktionen in der Datei `maintenance.log`.
Beim Einschalten werden in der Datei `maintenance.log` folgende Informationen gesucht:

- Wann wurde das letzte mal die Aktion "Bohnen nachfüllen" ausgeführt?
- Wann wurde das letzte mal die Aktion "Wasser nachfüllen" ausgeführt?
- Wann wurde das letzte mal die Aktion "entkalken" ausgeführt?
- Wie viele Aufgüsse wurden in Summe bereits getätigt?
- Wurde die Maschine korrekt abgeschalten? Wie viel Wasser und Bohnen enthielt sie zu dem Zeitpunkt? Wie viele Kaffees wurden seit dem letzten Entkalken gemacht?

Diese Informationen werden im Hauptmenü angezeigt. Können die Informationen im letzten Punkt (Wasser- und Bohnenmenge sowie die Anzahl an Kaffees seit dem Entkalken) nicht gefunden werden, werden die Standardwerte von Aufgabenblatt 2 (30g Bohnen und 200ml Wasser) verwendet.

Aufgabenstellung Die Klasse `CoffeeMachine` enthält bereits alle nötige Funktionalität, um die beschriebenen Anforderungen umzusetzen. Sie brauchen sich also um das korrekte Verhalten der Maschine, und die Verwaltung der Daten während des Betriebs, nicht zu kümmern. **Die Klasse `CoffeeMachine` darf nicht verändert werden.**
Ihre Aufgabe ist es, die von der Klasse aufgerufenen Funktionen zum Dateizugriff zu implementieren. Um Ihnen zu helfen sind zwei kleine Hilfsfunktionen bereits vorgegeben:

- `current_time() -> str` gibt den aktuellen Zeitstempel im korrekten Format zurück.
- `parse_coffee_line(line: str) -> int`: nimmt an, dass `line` ein Protokolleintrag zur Aktivität "Kaffee machen" ist, und gibt die darin enthaltene Anzahl der durchgeführten Aufgüsse an (extrahiert also die Zahl in runden Klammern).

Implementieren Sie die beiden folgenden Funktionen.
Für jede Funktion gilt:

1. Schließen Sie alle Dateien, die Sie geöffnet haben, wieder.
2. Beim Zugriff auf Dateien auftretende Exceptions müssen direkt in der jeweiligen Funktion abgefangen werden.

3. Geben Sie beim Abfangen einer Exception immer eine passende Nachricht mittels `print` aus, welche eine kurze eigene Fehlermeldung und die Meldung aus der Exception enthält.

Implementieren Sie diese Funktionen:

- `log_maintenance_task(task: str) -> str` fügt der Datei `maintenance.log` eine neue Zeile mit dem Inhalt `[<TIMESTAMP>]: task` hinzu, wobei `<TIMESTAMP>` die von `current_time()` zurückgegebene, aktuelle Zeit ist, und `task` der übergebene Parameter.

  Öffnen Sie die Datei so, dass ihr Inhalt erweitert wird, wenn sie bereits existiert, und sie neu erstellt wird, wenn sie noch nicht existiert.

  Die Funktion gibt immer den Wert von `<TIMESTAMP>` zurück — auch wenn beim Dateizugriff ein Problem aufgetreten ist.

- `read_log() -> dict` sucht in der Datei `maintenance.log`

  – Die jeweils letzte Zeile, welche ein Wasser nachfüllen/Bohnen nachfüllen/entkalken protokolliert. Dabei bezieht sich "letzte" auf die am weitesten "hinten" (oder "unten") in der Datei stehende Zeile. Wir überprüfen die Zeitstempel nicht, sondern nehmen an, dass die Einträge zeitlich geordnet in der Datei stehen.

    Es ist ausreichend die Zeilen dadurch zu identifizieren, dass sie die Strings `'beans refill'`, `'water refill'` bzw. `'decalcify'` enthalten. Der weitere Inhalt der Zeile muss nicht überprüft werden.

    Aus diesen Zeilen wird jeweils der Zeitstempel am Beginn (ohne Klammern) extrahiert. Dabei darf ohne weitere Überprüfung angenommen werden, dass jede solche Zeile mit dem String `[<TIMESTAMP>]` beginnt.

  – Alle Zeilen, in welchen ein "Kaffee machen" protokolliert wurde. Dazu ist es wiederum ausreichend, dass eine Zeile den String `'coffee made'` enthält. Aus diesen Zeilen wird mittels `parse_coffe_line()` die Anzahl der Aufgüsse extrahiert, und diese aufaddiert.

    Es ist wiederum nicht nötig, weiter zu überprüfen, ob die Zeile die definierte Struktur des Logeintrags erfüllt. Wenn eine Zeile den String `'coffee made'` enthält darf dies angenommen werden.

  – Die letzte nicht leere Zeile in `maintenance.log`. Enthält diese den String `'switched off'`, dann darf angenommen werden, dass es sich um eine Zeile der Form `[<TIMESTAMP>] switched off (water: <wasser>;beans: <bohnen>;decalc: <k>)` handelt. Aus dieser Zeile werden dann die Werte von `<wasser>`, `<bohnen>` und `<k>` – als `int` – extrahiert.

    Hinweis: Die Länge von `'[<TIMESTAMP>] switched off '` ist fix. Entfernen Sie zuerst diesen Teil der Zeile, sowie die Klammern `'()'`. Dann haben Sie mehrere Möglichkeiten den String zu zerlegen um die Zahlen zu extrahieren (entweder durch Verwendung von `str.split()` oder wiederholte Anwendung von `str.index()`). Nützen Sie anschließend, dass Sie die Längen der Strings `'water: '`, `'beans: '` und `'decalc: '` kennen.

  Die Funktion liefert immer ein Dictionary mit folgenden sieben Einträgen (keys) zurück: `'water refill'`, `'beans refill'`, `'last decalc'`, `'coffees made'`, `'water'`, `'beans'` und `'since decalc'`.

  Schlägt der Zugriff auf die Datei `maintenance.log` fehl (es wird eine Exception ausgelöst), dann enthalten `'water refill'`, `'beans refill'` und `'last decalc'` den String

`'read error'`, und alle anderen Einträge den Integerwert `-1`. Vergessen Sie auch die zuvor spezifizierten Anforderungen an die Behandlung von Exceptions nicht!

Zusatz (freiwillig): Versuchen Sie, durch das Abfangen der entsprechenden Exceptions in der richtigen Reihenfolge, zu unterscheiden, ob der Fehler beim Dateizugriff daran liegt, dass die Datei nicht gefunden werden konnte, oder ob ein anderer Fehler vorliegt. Ersetzen Sie die Strings `'read error'` durch `'no logfile'`, wenn die Datei nicht gefunden werden kann.

Kommt es bei dem Zugriff auf `maintenance.log` zu keiner Exception enthalten `'water refill'`, `'beans refill'` und `'last decalc'` den zuvor gefundenen, "letzten" Zeitstempel der jeweiligen Aktivität (als String), oder den String `'noch nie'`, falls es in der Datei dazu keinen Eintrag gibt. `'coffees made'` enthält die berechnete Summe an Kaffees (als `int`).

Die Werte von `'water'`, `'beans'` und `'since decalc'` hängen davon ab, ob die letzte Zeile der Datei eine `'switched off'` Zeile ist, oder nicht. Handelt es sich um eine `'switched off'` Zeile, dann enthalten `'water'`, `'beans'` und `'since decalc'` die aus der Zeile extrahierten Werte (als `int`). Sonst enthalten sie die Zahl `-1`.

Falls die letzte Zeile der Datei keine `'switched off'` Zeile ist, soll die Funktion außerdem mittels `print()` eine Meldung ausgeben, dass die Kaffeemaschine nicht ordnungsgemäß abgeschalten wurde.

Beispielaufrufe und erwartete Ergebnisse
`log_maintenance_task:`

Wenn die Datei `maintenance.log` nicht exisitert:
`log_maintenance_task('Test1')` gibt einen Timestamp `<TIMESTAMP>` der aktuellen Zeit zurück (z.B. `2025-05-02 16:10:53`). Anschließend existiert die Datei und enthält die Zeile

```
[<TIMESTAMP>]: Test1
```

Nach den anschließenden Aufrufen

```
log_maintenance_task('Test2')
log_maintenance_task('decalcify')
log_maintenance_task('beans refill (500g)')
log_maintenance_task('coffee made (2)[water: 150; beans: 30]')
log_maintenance_task('switched off (water: 50;beans: 450;decalc: 1)')
```

welche allen einen `<TIMESTAMP>` zurückliefern, enthält die Datei `maintenance.log` die Zeilen

```
[2025-05-02 16:05:04]: Test1
[2025-05-02 16:10:53]: Test2
[2025-05-02 16:10:53]: decalcify
[2025-05-02 16:10:53]: beans refill (500g)
[2025-05-02 16:10:53]: coffee made (2)[water: 150; beans: 30]
[2025-05-02 16:10:53]: switched off (water: 50;beans: 450;decalc: 1)
```

wobei hier korrekt formatierte Zeitangaben für `<TIMESTAMP>` eingesetzt wurden.

`read_log`:

Wird die Funktion aufgerufen, wenn die Datei `maintanance.log` nicht existiert, dann wird eine Fehlermeldung ausgegeben, z.B. `Fehler beim Lesen von maintenance.log:`
`[Errno 2] No such file or directory: 'maintenance_missing.log'`
und die Rückgabe der Funktion ist

```
{'water refill': 'read error', 'beans refill': 'read error',
 'last decalc': 'read error', 'coffees made': -1,
 'water': -1, 'beans': -1, 'since decalc': -1}
```

Dies ist auch das Verhalten, wenn es zu anderen Fehlern beim Zugriff auf die Datei kommt. Falls der beim Fehlen der Datei auftretende Fehler gesondert abgefangen wurde, wird eine spezielle Fehlermeldung ausgegeben, wie z.B. `Kann maintenance.log nicht finden:`
`[Errno 2] No such file or directory: 'maintenance_missing.log'`
und die Rückgabe der Funktion ist

```
{'water refill': 'no logfile', 'beans refill': 'no logfile',
 'last decalc': 'no logfile', 'coffees made': -1,
 'water': -1, 'beans': -1, 'since decalc': -1}
```

Zum weiteren Testen dieser Funktion gibt es in TUWEL ein Zip-Archiv mit verschiedenen Logdateien mit dem Namensschema `maintenance_<i>.log` für `<i>` $\in$ `{1, 2, 3, 4}`. Wird jeweils einer dieser Dateien in `maintenance.log` umbenannt, erzeugt der Aufruf `read_log()` folgende Rückgaben abhängig von der gewählten Datei:

`maintenance_1.log`:

```
{'water refill': '2025-04-16 15:15:43', 'beans refill': '2025-04-16 15:15:58',
 'last decalc': '2025-04-16 15:16:00', 'coffees made': 18, 'water': 764,
 'beans': 70, 'since decalc': 0}
```

`maintenance_2.log`:

```
{'water refill': '2025-04-01 12:00:21', 'beans refill': 'noch nie',
 'last decalc': 'noch nie', 'coffees made': 4, 'water': -1,
 'beans': -1, 'since decalc': -1}
```

`maintenance_3.log`:

```
{'water refill': 'noch nie', 'beans refill': 'GPA!!GPA!!GPA!!GPA!',
 'last decalc': 'noch nie', 'coffees made': 0, 'water': 120,
 'beans': 1200, 'since decalc': 17}
```

`maintenance_4.log`:

```
{'water refill': 'noch nie', 'beans refill': 'noch nie',
 'last decalc': 'noch nie', 'coffees made': 0, 'water': -1,
 'beans': -1, 'since decalc': -1}
```

Bei `maintenance_2.log` und `maintenance_4.log` wird außerdem noch eine Meldung ausgegeben, dass die Maschine nicht korrekt abgeschalten wurde, z.B.
`Kaffeemaschine wurde nicht korrekt abgeschalten.`

Kaffeemaschine:
Testen Sie zumindest folgende Abläufe:

- Starten Sie die Kaffeemaschine, ohne dass eine Datei `maintenance.log` existiert.

  Erwartetes Verhalten: Es gibt eine Ausgabe, dass `maintenance.log` nicht gefunden werden kann (aus `read_log()`). Die Datei wird angelegt und enthält zumindest die Zeile

  ```
  [2025-05-13 15:19:48]: switched on
  ```

- Starten Sie die Kaffeemaschine mit der Datei `maintenance_1.log` als `maintenance.log`:

  Erwaretetes Verhalten: Nach dem Starten sollte das Menü der Kaffeemaschine wie folgt aussehen:

  ```
  Genussvoller Profi Aufguss starting ... done
  ## Hallo, hier das Neueste von deiner Kaffeemaschine (ohne Pause seit <NOW>)
  === Status:
      Bohnen: 0.07 kg
      Wasser: 0.764 l

  === (Hauptmenü) Deine Möglichkeiten:
  [kaffee] Kaffee machen (already 18 made)
  [bohnen] Kaffeebohnen nachfüllen (last refill: 2025-04-16 15:15:58)
  [wasser] Wasser nachfüllen (last refill: 2025-04-16 15:15:43)
  [entkalken] Maschine entkalken (last: 2025-04-16 15:16:00)
  [quit] Maschine ausschalten
  ```

- Starten Sie die Kaffeemaschine (Existenz und Inhalt von `maintenance.log` beliebig), führen Sie ein paar Aktionen durch, und beenden Sie die Maschine mit dem Befehl `quit`.

  Erwartetes Verhalten: Überprüfen Sie die Datei `maintenance.log`. Wurden alle durchgeführten Aktionen korrekt protokolliert? Stimmen die Werte in der letzten Zeile (`'switched off'`) mit dem Endzustand der Maschine überein?

- Starten Sie die Kaffeemaschine erneut (mit der im vorherigen Durchlauf erzeugten Datei).

  Erwartetes Verhalten: Stimmen die im Menü angezeigten Informationen mit dem Zustand überein, in dem Sie die Maschine ausgeschalten haben?

- Starten Sie die Kaffeemaschine (Existenz und Inhalt von `maintenance.log` beliebig), führen Sie ein paar Aktionen durch, und beenden Sie das Programm ohne `quit` aufzurufen (brechen Sie die Python Ausführung ab, erzeugen Sie eine Exception, ...). Starten Sie die Kaffeemaschine erneut.

  Erwartetes Verhalten: Gab es beim Start eine Meldung, dass die Maschine nicht ordnungsgemäß beendet wurde? Zeigt die Kaffeemaschine zwar die korrekten Zeiten für das letzte mal Bohnen und Wasser nachfüllen an, aber die Standardmenge an Wasser und Bohnen?

Aufgabe 3 (Klassen (OOP)) [1 Punkt]

Ziel dieser Aufgabe ist es, ein Verständnis für die Idee und Funktionsweisen von Klassen und Objekten zu erhalten. Dazu sollen eigene Klassen definiert werden, Objekte von diesen Klassen erstellt und anschließend die Eigenschaften dieser Objekte manipuliert werden.
Dazu simulieren wir Restaurants, die Mittagsmenüs anbieten, und potentielle Gäste, welche die Angebote verschiedener Restaurants in Ihrer Nähe für die Mittagspause im Blick haben. Wir nehmen an, dass jedes Restaurant täglich ein Menü bietet.
Implementieren Sie dazu die folgenden Klassen. Achten Sie darauf, dass für jede Klasse die beschriebenen Attribute in der `__init__` Methode angelegt werden, und dass in den restlichen Methoden keine weiteren Attribute hinzugefügt werden.

Klasse Restaurant: Die Klasse repräsentiert Restaurants und besitzt die Attribute `name`, `menu_price`, `daily_portions`, `available_portions` und `cuisines`. Dabei enthält `name` den Namen des Restaurants, `menu_price` den Preis des Mittagsmenüs, `daily_portions` die Anzahl an Portionen, welche das Restaurant von seinem Menü täglich anbietet, `available_portions` die verbleibende Anzahl an Portionen des aktuellen Mittagmenüs, und `cuisines` ist eine Menge (set) welche die Namen (als Strings) der angebotenen Küchen (Italienisch, Chinesisch, ...) enthält.
Außerdem besitzt die Klasse folgende Methoden:

- `__init__`: besitzt die Parameter `name: str`, `menu_price: int` und `daily_portions: int`. Die Werte werden den entsprechenden Attributen des Objekts zugeordnet. Außerdem wird dem Attribut `available_portions` ebenfalls der Wert von `daily_portions` zugewiesen, und `cuisines` so initialisiert, dass es auf eine neue, leere Menge (set) verweist.

  Annahmen: `menu_price` ≥ 0, `daily_portions` ≥ 0.

- `reset_menus` setzt den Wert der verbleibenden Portionen (`available_portions`) zurück auf den Wert von `daily_portions`.

- `food_left` gibt `True` zurück, falls noch mindestens eine Portion des aktuellen Mittagmenüs übrig ist, und `False` sonst.

- `add_cuisine` erhält als Argument einen String `cuisine: str` und fügt diesen zur Menge der angebotenen Küchen hinzu.

- `adjust_price` erhält als Argument eine Zahl `change: int`. Ist die Summe des aktuellen Menüpreises mit dieser Zahl größer als 0, ändert die Methode den Menüpreis auf diese Summe und gibt `True` zurück. Andernfalls bleibt der Preis unverändert und die Methode gibt `False` zurück.

- `offers` erhält als Argument einen String `cuisine: str` und gibt `True` zurück, wenn das Restaurant diese Art der Küche anbietet, und `False` sonst.

- `sell_menus` erhält als Argument eine Zahl `amount: int`. Die Methode überprüft, ob das Restaurant noch eine entsprechende Anzahl an Portionen des Mittagmenüs besitzt. Ist dies der Fall werden `amount` viele Portionen verkauft, sonst so viele Portionen wie noch verfügbar sind. Die Methode reduziert die verfügbare Anzahl an Portionen um die verkaufte Anzahl, und gibt die verkaufte Anzahl zurück.

  Annahme: `amount` ≥ 0

- `print_status` gibt mittels `print()` eine Übersicht des Angebots des Restaurants aus. Sie können die Ausgestaltung der Ausgabe frei wählen, sie muss jedoch zumindest die folgenden Informationen enthalten: Den Namen des Restaurants, und, falls noch Portionen übrig sind, die Anzahl an verbleibenden Portionen, den Preis des Menüs, und eine Auflistung der gebotenen Küchen. Sind keine Portionen mehr übrig muss statt den Informationen zu den Essen ein Hinweis ausgegeben werden, dass die Menüs für heute ausverkauft sind.

Klasse Foodie: Die Klasse besitzt die Attribute `name: str`, `budget: int`, `default_amount: int` und `watchlist: set[Restaurant]`, wobei `name` der Name der Person ist, `budget` ihr tägliches Budget für Mittagessen darstellt, `default_amount: int` die typische Anzahl an Portionen, die sie pro Tag isst, und `watchlist` eine Menge (set) von `Restaurant` Objekten, welche die Person als mögliche Quelle für ein Mittagessen im Auge behält.
Außerdem besitzt die Klasse folgende Methoden:

- `__init__`: besitzt die Parameter `name: str`, `budget: int` und `default_amount: int`, und weist die Werte der Parameter dem passenden Attribut zu. Außerdem wird das Attribut `watchlist` als leere Menge (set) initialisiert.

- `watch_restaurant` erhält als Argument ein `Restaurant`-Objekt `restaurant: Restaurant`, welches zur Watchlist der Person hinzugefügt wird.

- `increase_budget` besitzt keine Parameter und erhöht das Budget der Person um 2.

- `buy_menus` besitzt zwei Parameter: ein `Restaurant`-Objekt `restaurant: Restaurant`, bei welchem Essen gekauft werden soll, und eine Zahl `amount: int` mit dem Default Wert von `-1`. Ist `amount` kleiner als `-1`, gibt die Funktion `False` zurück. Andernfalls bestimmt der Wert von `amount` die Anzahl an Portionen, die versucht werden bei `restaurant` zu kaufen:

  Hat `amount` den Wert `-1`, dann werden `default_amount` viele Portionen gekauft, ansonsten `amount` viele Portionen. Die Funktion versucht, die entsprechende Anzahl an Portionen zu kaufen (durch Aufruf von `sell_menus` in `restaurant`), und gibt `True` zurück, wenn dies gelingt, und `False`, wenn dabei nur eine geringere Anzahl an Portionen gekauft werden konnten.

- `list_options` besitzt einen Paramter `cuisine: str` mit dem Default-Wert `None`. Die Funktion erzeugt mittels `print()` eine Ausgabe. Deren genaue Ausgestaltung ist Ihnen wiederum frei gestellt, sie muss jedoch zumindest folgende Informationen und Verhalten aufweisen: Die Ausgabe erstellt einen Bericht für die Person mit der ihr zur Verfügung stehenden Optionen. Sie enthält auf jeden Fall den Namen der Person. Anschließend folgt eine Auflistung aller Restaurants in der `watchlist` der Person, welche die in `cuisine` spezifizierte Küche anbieten. Besitzt `cuisine` den Wert `None`, dann werden alle Restaurants in der `watchlist` ausgegeben. Zu jedem ausgegebenen Restaurant werden folgende Informationen angegeben:

  - Eine Zeile mit dem Namen des Restaurants, sowie einem Zusatz, falls die Kosten eines Menüs über dem Budget der Person liegen (eine Warnung/ein Hinweis, dass das Restaurant zu teuer ist).
  - Abhängig davon, ob das Restaurant schon ausverkauft ist oder nicht entweder eine Zeile mit der Information, dass das Essen ausverkauft ist, oder mit der Anzahl der noch verfügbaren Portionen und dem Preis eines Menüs.

Beispielaufrufe und (mögliche) Ergebnisse:

```
gpa = Restaurant('Grob Püriertes Aufgetautes', 5, 6)
gpa.print_status()                                                    gibt aus:
                  ** Grob Püriertes Aufgetautes **
                   Menüs übrig: 6
                   Preis: 5
                   Wir bieten:
```

Anschließende Aufrufe

```
gpa.offers('Italienisch')                                  gibt False zurück

gpa.add_cuisine('Chinesisch')          hat keine explizite Rück- oder Ausgabe
gpa.add_cuisine('Libanesisch')         hat keine explizite Rück- oder Ausgabe
gpa.offers('Italienisch')                                  gibt False zurück
gpa.offers('Chinesisch')                                    gibt True zurück
gpa.sell_menus(2)                                              gibt 2 zurück

gpa.print_status()                                                    gibt aus:
                  ** Grob Püriertes Aufgetautes **
                   Menüs übrig: 4
                   Preis: 5
                   Wir bieten: Libanesisch, Chinesisch
gpa.adjust_price(-6)                                       gibt False zurück
gpa.adjust_price(1)                                         gibt True zurück
gpa.print_status()                                                    gibt aus:
                  ** Grob Püriertes Aufgetautes **
                   Menüs übrig: 4
                   Preis: 6
                   Wir bieten: Libanesisch, Chinesisch
gpa.food_left()                                             gibt True zurück
gpa.sell_menus(5)                                              gibt 4 zurück
gpa.food_left()                                            gibt False zurück
gpa.reset_menus()                      hat keine explizite Rück- oder Ausgabe
gpa.print_status()                                                    gibt aus:
                  ** Grob Püriertes Aufgetautes **
                   Menüs übrig: 6
                   Preis: 6
                   Wir bieten: Libanesisch, Chinesisch
```

Das Erstellen einer Person: (im Anschluss an obige Aufrufe)

```
obelix = Foodie('Obelix', 5, 3)
obelix.list_options()                                          gibt aus:
                Hallo Obelix!
                Das sind deine Optionen:
obelix.watch_restaurant(gpa)                  hat keine explizite Rück- oder Ausgabe
obelix.watch_restaurant(gpa)                  hat keine explizite Rück- oder Ausgabe
obelix.list_options()                                          gibt aus:
                Hallo Obelix!
                Das sind deine Optionen:
                == Grob Püriertes Aufgetautes: **ZU TEUER**
                ** 6 Menüs um 6 Euro verfügbar.
obelix.buy_menus(gpa, 1)                                  gibt True zurück
obelix.buy_menus(gpa)                                     gibt True zurück
obelix.buy_menus(gpa)                                    gibt False zurück
obelix.list_options()                                          gibt aus:
                Hallo Obelix!
                Das sind deine Optionen:
                == Grob Püriertes Aufgetautes: **ZU TEUER**
                 ** AUSVERKAUFT **
obelix.increase_budget()                      hat keine explizite Rück- oder Ausgabe
obelix.list_options()                                          gibt aus:
                Hallo Obelix!
                Das sind deine Optionen:
                == Grob Püriertes Aufgetautes:
                 ** AUSVERKAUFT **
gpa.reset_menus()                             hat keine explizite Rück- oder Ausgabe
wurst = Restaurant('Mizitants Würstelstand', 2, 12)
obelix.watch_restaurant(wurst)                hat keine explizite Rück- oder Ausgabe
wurst.add_cuisine('Wurst')
obelix.list_options('Libanesisch')                             gibt aus:
                Hallo Obelix!
                Das sind deine Optionen:
                == Grob Püriertes Aufgetautes:
                ** 6 Menüs um 6 Euro verfügbar.
obelix.list_options('Wurst')                                   gibt aus:
                Hallo Obelix!
                Das sind deine Optionen:
                == Mizitants Würstelstand:
                ** 12 Menüs um 2 Euro verfügbar.
```

In der Angabedatei `blatt6.py` finden Sie einen möglichen Programmablauf in der Funktion `lunch_scenario()` gegeben. Zusätzlich sind die erwarteten Ausgaben als Kommentar angegeben.

Aufgabe 4 (O-Notation) [1 Punkt]

Ziel dieser Aufgabe ist es, mit dem Konzept der O-Notation vertraut zu werden. Dies umfasst die Entscheidung ob $f(n)$ *in* $O(g(n))$ für zwei Funktionen f und g gilt, sowie das Bestimmen entsprechender oberer Schranken. Die Aufgabe besteht aus zwei Teilaufgaben:

1. In der ersten Teilaufgabe ist jeweils eine Funktion und ein Ausdruck in O-Notation gegeben. Bestimmen Sie für jede Funktion, ob der Ausdruck in O-Notation eine asymptotische obere Schranke der Funktion darstellt.

   Geben Sie in Ihrer Abgabedatei in der Zeile `Antwort` an, ob dies der Fall ist.

   Tragen Sie im positiven Fall in der Zeile `Begründung` Werte für n_0 und c ein, welche die Beziehung beweisen. Diese müssen nicht die kleinsten möglichen sein.

   Tragen Sie im negativen Fall in der Zeile `Begründung` eine intuitive Erklärung ein, warum die Beziehung nicht gilt (max. 1–2 Sätze).

   - Ist $2^n - n^5 + 2 \cdot n^3 - 2025$ in $O(n^6)$?
   - Ist $n^3 + 2^{10} \cdot n$ in $O(n^4)$?
   - Ist $n \cdot \log^2(n) + 20^4$ in $O(n^3)$?
   - Ist $\frac{n^2}{n} + n^2 \cdot n - 30 \cdot n$ in $O(n^2)$?

   In der Python-Datei zur Angabe ist bereits ein Beispiel vorgegeben.

2. Im zweiten Teil der Aufgabe ist jeweils nur eine Funktion gegeben. Bestimmen Sie für jede Funktion die **asymptotisch kleinste obere Schranke**. Geben Sie die Schranke dabei **so einfach wie möglich** an (vgl. Vorlesungsfolien "Algorithmen – Einführung", Folie 42ff).

   Geben Sie die Schranke wiederum in Ihrer Abgabedatei an, gemeinsam mit Werten für n_0 und c, welche belegen, dass es sich hierbei wirklich um eine asymptotische obere Schranke handelt.

   - $5 \cdot n \cdot \log(n+3) - 12 \cdot n + 6$
   - $2^3 \cdot n \cdot n + 42 - 8 \cdot n^2$
   - $14 \cdot n^3 + 10^5 \cdot \log_2(n) + 2 \cdot n^2 \cdot 2 \cdot n^2$

   In der Python-Datei zur Angabe ist wiederum ein Beispiel vorgegeben.

Hinweis (für beide Unteraufgaben): Die Werte für n_0 und c müssen nicht die kleinstmöglichen Werte sein – sie müssen nur die Anforderungen der Definition erfüllen. Es muss außerdem nicht gezeigt werden, dass die Werte korrekt sind. Das Angeben richtiger Werte reicht aus.