Python – Testat 1 (FS2022)

1 Aufgabenstellung und Bewertungskriterien

☞ Implementieren Sie die Klassen `ModuleBuilder` und `MissingModuleError`, wie unten im Detail beschrieben und liefern Sie diese **als einzelne Datei mit dem Namen** `testat1_<PId>.txt`[1] der Dozierenden **bis spätestens am 26.04.2022 um 12 Uhr per Email** (an selina.malacarne@ost.ch) ab.
Dieser Testataufgabe ist ein **gezippter Ordner beigefügt**, welcher alle **für die Aufgabenstellung relevanten Informationen** enthält, darin finden Sie auch ein `testat1_<PId>.py`-File, welches Sie **für die Erarbeitung Ihrer Lösung verwenden sollen**.

Bewertungskriterien: Für jedes Detail, das richtig implementiert wird, gibt es Punkte.
Rechts neben jeder Detailbeschreibung, wird die Zahl der möglichen Punkte angegeben.
In diesem Testat 1 können insgesamt **20 Punkte** geholt werden.
Die unten **angegebenen Namen von Klassen, Methoden, Attribute, Parameter und Keys müssen exakt übernommen** werden, sonst werden für den betroffenen Teil keine Punkte vergeben, denn **Ihre Lösung wird mit einer "Prüferklasse" automatisch getestet**.
Wenn die vorgegebenen Namen nicht stimmen, dann kann die Prüferklasse diese nicht aufrufen und vergibt entsprechend keine Punkte. Es wird daher empfohlen, den eigenen Code ausgiebig zu testen. Die Code-Beispiele in den grauen Boxen eignen sich für einen ersten Vergleich.
Direkt unterhalb der Zahl der möglichen Punkte finden Sie die Testnummern (bspw. T201, T202 etc.). Diese Nummern repräsentieren die automatischen Tests, welche mit der Prüferklasse geprüft werden. Sie erhalten nach Abgabe Ihrer Lösung die Auswertung in Form eines Textfiles mit einer Auflistung der Ergebnisse eines jeden Tests.
Beiliegend zu dieser Aufgabenstellung erhalten Sie ein Beispiel eines solchen Auswertungsfiles. Darin sehen Sie noch etwas detaillierter, was genau in den einzelnen Tests geprüft wird und welche Teilpunkte für jeden erfolgreichen Test vergeben werden.

Prüfungszulassung: Um an der Prüfung zugelassen zu werden, muss **mindestens 40% der erreichbaren Punkte pro Testat 1 & 2 erreicht** werden.

[1] *.py Dateien werden vom OST Mailserver blockiert, ihre PId können Sie aus der Python Klassenliste vom Unterrichtsportal entnehmen.

2 Allgemeine Bewertungskriterien

- **Nur die unten aufgelisteten *public* Methoden und Variablen dürfen *public* sein**. Sollten Sie weitere Attribute benötigen, so müssen diese entweder *protected* oder *private* sein. 1 P. T201 T202
- Die Klassen und alle zu implementierenden Methoden (`__init__()`, `__str__()`, etc.) besitzen aussagekräftige Docstrings. 1 P. T203
- Der Python-Codestil entspricht den PEP8-Empfehlungen.[2] 1 P. T204

2.1 Nach PEP8-Stilfehlern suchen

Das Flake8-Programm kann wie folgt installiert werden:

1. "Anaconda Prompt" **als Administrator öffnen** und folgende Zeile ausführen:
2. `conda install -c anaconda flake8`

Überprüfen Sie ihren Python-Code nach Stilfehlern, indem Sie das Flake8-Programm[3] wie folgt in der "Anaconda Prompt" aufrufen:

```
flake8 python_datei.py
```

[2] Der Codestil wird mit der Flake8-Software geprüft, `http://flake8.pycqa.org/en/latest/`

[3] Die Anaconda Prompt muss beim Filepfad des zu prüfenden Pythonskripts geöffnet sein. Zum Filepfad navigieren Sie mit dem Befehl `cd <path>`

3 Einleitung

Eine Firma produziert und vertreibt modulare Monitoring Systeme, welche in der Lage sind verschiedene Zustandsgrössen wie die Luftfeuchtigkeit, den CO2-Gehalt oder die relative Feuchte zu messen. Je nach Anwendung werden hierbei verschiedene Arten von Sensormodulen und zusätzliche Module benötigt. Die Firma bietet den Kunden daher die Möglichkeit, ihr Monitoring System selbstständig über den Webshop zusammenzustellen und zu bestellen.
Um ein vom Kunden zusammengestelltes Monitoring System produzieren zu können, wird in der Produktion eine Produktionsliste benötigt, welche alle relevanten Informationen über die zum System gehörigen Module gespeichert hat. Zu diesen Informationen gehören unter anderem die Bezeichnungen der Module (Sensoren, Controller, Anzeigen) mit entsprechenden Referenzen für die Produktion sowie die gesamte Anzahl zu produzierender Einheiten des Systems.
Diese Liste wird direkt aus dem Webshop an die Produktion weiter geleitet und von der Produktionssoftware für die weitere Verwendung vorbereitet. Die nachfolgende Tabelle zeigt ein Beispiel einer solchen Produktionsliste:

Number of Units	2	
Reference	Article Number	Module Description
T1	TF1902	Temperature Sensor
T2	TF1902	Temperature Sensor
C1	CO304	CO2 Sensor
F1	HL20D	Humidity Sensor
P1	P394N	Pressure Sensor
M1	MCU940T	Microcontroller Module
D1	DP419	Display Module
G1	AL130	Enclosure

Abbildung 1: Beispiel einer Produktionsliste aus dem Webshop.

Bevor die bestellten Systeme produziert werden können, müssen natürlich die entsprechenden Module aus dem Lager bezogen werden. In der folgenden Testataufgabe soll die Klasse `ModuleBuilder` implementiert werden, welche zur Erstellung der vom Kunden bestellten Monitoring Systeme verwendet werden kann. Des Weiteren soll die Klasse `MissingModuleError` implementiert werden, welche eine für die Anwendung passende Exception implementiert.

4 Klasse `Storage`

Diese Klasse verwaltet das Lager, in welcher diverse Module zu finden sind. Normalerweise wären die Lagerdaten aller im Lager befindlichen Module in einer Datenbank auf einem externen Server gespeichert. Für diese Aufgabe handelt es sich beim Lager um einen lokalen Ordner auf ihrem Computer, in welchem `*.txt` Files abgelegt sind.
Jedes Modul (bzw. jede Artikelnummer) erhält dabei sein eigenes `<article_nummer>.txt` File in diesem Ordner, in welchem die Anzahl der im Lager befindlichen Module sowie zusätzliche Informationen über das Modul gespeichert sind. Das folgende Beispiel zeigt den Inhalt eines solchen Files anhand des Moduls mit Artikelnummer `CO304`:

```
# Inhalt des Files CO304.txt
count=49
description=C02 sensor
accuracy=+/-30ppm
measuring_range=0...5000ppm
power_supply=20...40VDC/14...28VAC
interface=USB/UART/I2C
weight=99g
price=79.95
currency=CHF
```

Auf die im Lager befindlichen Module kann mit der abgegebenen `Storage`-Klasse zugegriffen werden. Die **bereits fertig implementierte** Klasse finden Sie im mit der Aufgabenstellung abgegebenen Ordner im File `storage_lib.py`.
Im File `testat1_<PId>.py`, in welchem Sie ihre Lösung implementieren sollen, ist die Klasse über folgende Zeile bereits importiert:

```
from storage_lib import Storage
```

Bei der Instanziierung eines `Storage`-Objekts muss lediglich der Pfad auf den `"Storage"` Ordner als `str` angegeben werden damit die Klasse Zugriff auf die Datenbank-Daten erhält, wie in folgendem Beispiel gezeigt:

```
>>> from storage_lib import Storage
>>> s = Storage("D:/testat1/storage")
```

Die Klasse hat folgende **zwei public Methoden**:

- `Storage.module_info(self, article_number)`
  Mit dieser Methode kann man die Informationen eines im Lager befindlichen Modules abfragen, indem man ihr die Artikelnummer des gewünschten Modules in Form eines `str` übergibt.
  Als Rückgabewert erhält man ein **Dictionary**, in welchem alle Informationen über das Modul gespeichert sind. Die im Dictionary gegebenen Informationen können von Modultyp zu Modultyp verschieden sein. Das Dictionary für das Modul mit Artikelnummer `CO304` sieht bspw. wie folgt aus:

```
>>> print(s.module_info("CO304"))
{"count": "49",
 "description": "CO2 sensor",
 "accuracy": "+/-30ppm",
 "measuring_range": "0...5000ppm",
 "power_supply": "20...40VDC/14...28VAC",
 "interface": "USB/UART/I2C",
 "weight": "99g",
 "price": "79.95",
 "currency": "CHF"}
```

  Wird der Methode eine unbekannte Artikelnummer übergeben, welche nicht im Lager vorhanden ist, wird `None` zurückgegeben:

```
>>> print(s.module_info("xyz"))
None
```

- `Storage.take_module(self, article_number, count)`
  Mit dieser Methode kann man Module aus dem Lager entnehmen. Dazu muss man die gewünschte Artikelnummer und die Anzahl angeben.
  Sind genügend Module im Lager vorhanden, so gibt die Methode die gewünschte Anzahl und die Artikelnummer als **Tupel** zurück:

```
>>> print(s.take_module("CO304", 2))
(2, "CO304")
```

  Sind nicht genügend oder gar keine Module einer bestimmten Artikelnummer im Lager vorhanden, so gibt die Methode `None` zurück:

```
>>> print(s.take_module("CO304", 2000))
None
```

5 Klasse `ModuleBuilder`

Diese Klasse ist für die Produktion der vom Kunden gewünschten Monitoring Systeme zuständig. Die Bestellung des Kunden erfolgt automatisch über den Webshop und ist dann in Form einer Produktionsliste gegeben. Sie finden Beispieldateien solcher Produktionslisten (zu Testzwecken) im beigefügten Ordner `orders`, wie z.B.:

```
Number of Units;2;
Reference;Article Number;Module Description
T1;TF1902;Temperature Sensor
T2;TF1902;Temperature Sensor
C1;CO304;CO2 Sensor
F1;HL20D;Humidity Sensor
P1;P394N;Pressure Sensor
M1;MCU940T;Microcontroller Module
D1;DP419;Display Module
G1;AL130;Enclosure
```

5.1 Methode `__init__(self, order_file, storage_path)` (4 P)

Die Methode nimmt neben dem Parameter `self` die beiden notwendigen Parameter `order_file` und `storage_path` entgegen und initialisiert die folgenden **drei protected Instanzvariablen**:

- `self._order_data`
  Ein **Dictionary**, in welchem alle Informationen aus der aktuell anstehenden Produktionsliste gespeichert werden. Der Pfad des Files wird mit dem Parameter `order_file` bei der Instanziierung übergeben und referenziert eine Produktionsliste im Aufbau gleich wie es in der Einleitung gezeigt worden ist.
  Die Keys im Dictionary sind jeweils die in der Produktionsliste angegebenen Artikelnummern (`TF1902`, `CO304`, `HL20D`, etc.). Jedem Key ist als Value eine **Liste** zugeordnet, welche alle zur Artikelnummer zugehörigen Referenzen speichert (`T1, T2, C1, F1` etc.). Alle restlichen Informationen in der Produktionsliste werden ignoriert.
  Lesen Sie das File `order_file` mit `encoding="utf-8"`.

  2 P. T511 T512 T513

- `self._units`
  In dieser Instanzvariable soll die Anzahl zu produzierender Einheiten als **int** gespeichert werden. Diese Zahl ist ebenfalls in der Produktionsliste (im Pfad `order_file`) zu finden und zwar auf der ersten Zeile (nach `"Number of Units"`).

  1 P. T514 T515

- `self._storage`
  Ein Objekt der Klasse Storage, mit welchem über die Methoden `modul_info(article_number)` und `take_module(article_number, count)` auf das Lager zugegriffen werden kann. Bei der Instanziierung des `Storage` Objektes `self._storage` muss der Pfad zum Ordner mit den Lagerdaten übergeben werden, welcher durch den Parameter `storage_path` gegeben ist. Weitere Informationen zur `Storage` Klasse finden Sie in Abschnitt 4.

  1 P. T516

Die Instanziierung eines `ModuleBuilder`-Objektes kann wie folgt lauten:

```
>>> mb = ModuleBuilder("D:/testat1/orders/produktionsliste_1.csv", "D:/testat1/storage")
```

Für die in der Einführung gezeigten Produktionsliste (siehe mitgelieferter Ordner `testat1/orders/produktionsliste_1.csv`) sieht der Inhalt des Dictionary `self._order_data` nach der Instanziierung wie folgt aus:

```
>>> print(mb._order_data)
{"TF1902": ["T1", "T2"],
 "CO304": ["C1"],
 "HL20D": ["F1"],
 "P394N": ["P1"],
 "MCU940T": ["M1"],
 "DP419": ["D1"],
 "AL130": ["G1"]}
```

Die protected Variable `self._units` hat die zu produzierende Anzahl aus der Produktionsliste gespeichert:

```
>>> print(mb._units)
2
```

Die protected Variable `self._storage` referenziert ein Objekt der `Storage` Klasse, welche im `storage_lib.py` File implementiert ist:

```
>>> print(type(mb._storage))
<class "storage_lib.Storage">
```

5.2 Methode `__str__(self)` (3 P)

3 P.
T521
T522
T523

Mit der magischen Methode soll der Inhalt der Instanzvariable `self._order_data` (und die Anzahl der in `self._order_data` gegebenen Referenzen) in Form eines hübsch formatierten Strings zurückgegeben werden.
Der String besteht aus drei Spalten: Article Number, Quantity, References.
Die Reihenfolge der aufgelisteten Module soll anhand der Artikelnummer alphabetisch aufsteigend sein. Die Daten sollen wie die Spaltenbeschriftung linksbündig ausgerichtet sein.
Zwischen den Spalten sollen 4 Leerzeichen platziert werden. Beachten Sie, dass in der letzten Spalte jeder Eintrag, der kürzer als der Eintrag mit der maximalen Spaltenbreite ist, mit Leerzeichen aufgefüllt ist. Die Trennung der Spaltenbeschriftung und deren Inhalt erfolgt über eine Aneinanderreihung von Minuszeichen, und zwar genau so viele an der Zahl wie es gesamte Anzahl Zeichen in der breitesten Zeile hat.
Jede Spaltenbreite passt sich automatisch anhand des längsten Eintrags in dieser Spalte an.
Wird ein `ModuleBuilder`-Objekt der `print()`-Funktion übergeben, so sieht der erzeugte String wie folgt aus:[4]

```
>>> print(mb)
Article Number    Quantity    References
----------------------------------------
AL130             1           G1
CO304             1           C1
DP419             1           D1
HL20D             1           F1
MCU940T           1           M1
P394N             1           P1
TF1902            2           T1,T2
```

Nachfolgend ist ein weiteres Beispiel gezeigt, bei welcher eine Produktionsliste geladen wird, für deren Daten die Spaltenbreite automatisch angepasst wird:

```
>>> mb2 = ModuleBuilder("D:/testat1/orders/produktionsliste_2.csv",
   "D:/testat1/storage")
>>> print(mb2)
Article Number    Quantity    References
-------------------------------------------
AL130             1           G1
CO304             1           C1
DP419             1           D1
HL20D             1           F1
MCU940T           1           M1
P394N             1           P1
TF1902            5           T1,T2,T3,T4,T5
```

[4]Die Referenzen `T1,T2` sind nur durch ein Komma separiert, es hat keine Leerschläge dazwischen.

5.3 Methode `availability(self, units=None)` (4 P)

Diese Methode erstellt eine Übersicht zur Verfügbarkeit aller für die Produktion benötigten Module (gegeben durch `self._order_data`) in Form eines Dictionaries, und zwar anhand der Daten aus dem Lager `self._storage` und der Instanzvariable `self._units` (sofern der Parameter `units` gleich `None` ist), welche die Anzahl zu produzierender Einheiten repräsentiert. Die Methode nimmt folgenden Parameter entgegen:

`units`	Der Parameter gibt dem Anwender die Möglichkeit eine andere Anzahl zu produzierender Einheiten anzugeben. Der Defaultwert ist `None`.

- Die Methode liefert anhand der Daten im Dictionary `self._order_data`, der Lagerdaten `self._storage` sowie der Variable `self._units` (sofern `units` gleich `None` ist) ein **Dictionary** von Dictionaries in folgender Form zurück: (3 P. T531 T532 T533 T534 T535 T536)

```
{"TF1902": {"available": 100, "required": 4, "missing": 0},
 "CO304": {"available": 0, "required": 2, "missing": 2},
 "HL20D": ...
 ...
}
```

  Für jede `article_number` in `self._order_data` wird im oben gezeigten Dictionary ein Dictionary mit den Schlüsseln `"available"`, `"required"` und `"missing"` erstellt, wobei die Schlüssel-Wert-Paare wie folgt zu interpretieren sind:

  `"available"`: `available`
  Die im Lager befindliche Anzahl Module (anhand der `count`-Information im Dictionary `self._storage_data`) des entsprechenden Typs `article_number`.

  `"required"`: `required`
  Die benötigte Anzahl dieses Modultyps aus der Produktionsliste `self._order_data` multipliziert mit der Anzahl gewünschter Einheiten.
  Ist der Parameter `units` gleich `None`, so wird mit dem Wert der Instanzvariable `self._units` gerechnet, ansonsten mit dem gegebenen Parameter `units`.
  Der durch `units` gegebene Wert muss `> 0` sein. Ist dies nicht der Fall, so soll eine `ValueError()`-Exception geworfen werden, mit der folgenden Nachricht:
  `"The given value for units must be greater than 0."`.
  Beim Wert `self._units` muss keine Prüfung `> 0` erfolgen, bei diesem Wert kann davon ausgegangen werden, dass er korrekt ist.

  `"missing"`: `missing`
  Die fehlende Anzahl Module des entsprechenden Typs `article_number`, d.h. die Differenz aus `"required"` und `"available"`.

  Die angegebenen Mengen `available`, `required`, `missing` sind vom Typ `int`.

- Sollte eine für die Produktion benötigte `article_number` nicht im Lager `self._storage` vorhanden sein, wird im zurückgegebenen Dictionary beim Schlüssel der entsprechenden `article_number` der Wert `missing` auf den Wert `required` gesetzt: 1 P. T537

```
{article_number: {"available": 0, "required": 55, "missing": 55}, ...}
```

Der Aufruf der Methode kann beispielsweise wie folgt aussehen:

```
# check availability
>>> print(mb.availability())
{"TF1902": {"available": 100, "required": 4, "missing": 0},
 "CO304": {"available": 0, "required": 2, "missing": 2},
 "HL20D": {"available": 150, "required": 2, "missing": 0},
 "P394N": {"available": 75, "required": 2, "missing": 0},
 "MCU940T": {"available": 0, "required": 2, "missing": 2},
 "AL130": {"available": 17, "required": 2, "missing": 0}}
```

Der Aufruf der Methode mit einer korrekten Angabe für den Parameter `units`:

```
# check availability
>>> print(mb.availability(units=10))
{"TF1902": {"available": 100, "required": 20, "missing": 0},
 "CO304": {"available": 0, "required": 10, "missing": 10},
 "HL20D": {"available": 150, "required": 10, "missing": 0},
 "P394N": {"available": 75, "required": 10, "missing": 0},
 "MCU940T": {"available": 0, "required": 10, "missing": 10},
 "AL130": {"available": 17, "required": 10, "missing": 0}}
```

Der Aufruf der Methode mit einer falschen Angabe für den Parameter `units`:

```
# check availability
>>> print(mb.availability(units=-10))
...
ValueError: The given value for units must be greater than 0.
```

5.4 Methode `build(self)` (3 P)

Diese Methode erstellt die durch `self._units` gegebene Anzahl Einheiten des vom Kunden bestellten Monitoring Systems, welches im Dictionary `self._order_data` zusammengefasst ist. Die Methode nimmt – ausser der Referenz auf das Objekt – keine weiteren Parameter entgegen.

- Die Methode prüft zuerst, ob alle für die zu produzierenden Einheiten benötigten Module (gegeben durch `self._units` und `self._order_data`) im Lager `self._storage` vorhanden sind. Sollte ein oder mehr Module fehlen, so soll eine `MissingModuleError`-Exception geworfen werden, welche im Abschnitt 6 näher beschrieben ist.
  Der Exception soll eine Liste übergeben werden, welche die Artikelnummern aller fehlender Module als `str` auflistet.
  Tipp: Benutzen Sie die Methode `self.availability(self, units=None)`, um die Verfügbarkeit der einzelnen Module zu prüfen.
  In folgendem Beispiel wird versucht die gewünschten Einheiten mit `build()` zu erzeugen, jedoch hat es von vier Modulen keine oder zu wenige im Lager:

  2 P.
  T541
  T542

```
>>> mb.build()
...
MissingModuleError: The following modules are missing from storage: TF1902,
    HL20D, P394N, AL130
```

- Wenn keine Bauteile fehlen, dann sollen die für die Produktion benötigten Module in der benötigten Anzahl aus dem Lager entnommen werden.
  Um die entsprechende Anzahl eines Artikels aus dem `self._storage` zu entnehmen, soll die `take_module(self, article_number, count)` der `Storage`-Klasse verwendet werden, dabei wird die Anzahl Module im Lager automatisch durch die entnommene Anzahl dekrementiert. Die Methode `build()` soll schlussendlich die gesamte Anzahl aus dem Lager entnommenen Module zurückgeben, wie im folgenden Beispiel gezeigt ist:

  1 P.
  T543
  T544

```
>>> mb.build()
16
```

6 Klasse `MissingModuleError`

Diese Klasse erbt von der `Exception`-Klasse und kann dafür verwendet werden, dem Produktionsmitarbeiter mitzuteilen, dass für die Fertigung benötigte Module im Lager nicht verfügbar sind.

6.1 Methode `__init__(self, modules)` (3 P)

Wird eine `MissingModuleError`-Exception geworfen, so muss ihr der folgende Parameter übergeben werden:

`modules`	Eine Liste der Artikelnummern (vom Typ `str`) aller im Lager fehlender Module.

Die Klasse hat folgende public Instanzvariable:

- `self.modules`
  Der Parameter `modules` soll zu einem **Tupel** umgewandelt und in der public Instanzvariablen `self.modules` gespeichert werden. (1 P. T611 T612)

- Des Weiteren soll eine für die Exception passende Message erzeugt werden, welche immer dann ausgegeben wird, wenn die Exception geworfen wird. Diese Message ist von folgender Form:
  `"The following modules are missing from storage: {x}"`, wobei mit `{x}` die im Tupel `self.modules` gegebenen Artikelnummern gemeint sind, welche kommasepariert angegeben werden sollen. Die Message soll nach der Erzeugung nicht in einer Instanzvariable gespeichert werden sondern direkt der `__init__()`-Methode der Superklasse übergeben werden, welche die Ausgabe auf die Konsole nach der Initialisierung automatisch übernimmt. (2 P. T613)

Im folgenden Beispiel wird zu Testzwecken eine `MissingModuleError`-Exception geworfen und als Argument wird eine Liste von Artikelnummern angegeben, welche dann kommasepariert ausgegeben werden:[5]

```
>>> raise MissingModuleError(["TF1902", "HL20D", "P394N"])
...
MissingModuleError: The following modules are missing from storage: TF1902, HL20D, P394N
```

Im folgenden Beispiel wird versucht die gewünschten Einheiten mit `build()` zu erzeugen, jedoch hat es von vier Modulen keine oder zu wenige im Lager:

```
>>> mb.build()
...
MissingModuleError: The following modules are missing from storage: TF1902, HL20D,
    P394N, AL130
```

Dasselbe Beispiel noch einmal, hier wird die Exception jedoch abgefangen und nur die Instanzvariable `modules` betrachtet:

```
>>> try:
>>>     print(mb.build())
>>> except MissingModuleError as e:
>>>     print(e.modules)
("TF1902", "HL20D", "P394N", "AL130")
```

[5] Nach jedem Komma steht hier noch ein Leerschlag.