

Python – Testat 1

1 Einleitung

Bei der Fertigung von elektronischen Leiterplatten (engl. PCB für Printed Circuit Boards) müssen diese nicht nur hergestellt sondern anschliessend auch noch mit den entsprechenden Bauteilen bestückt werden. Um das PCB bestücken zu können, benötigt man eine Stückliste,

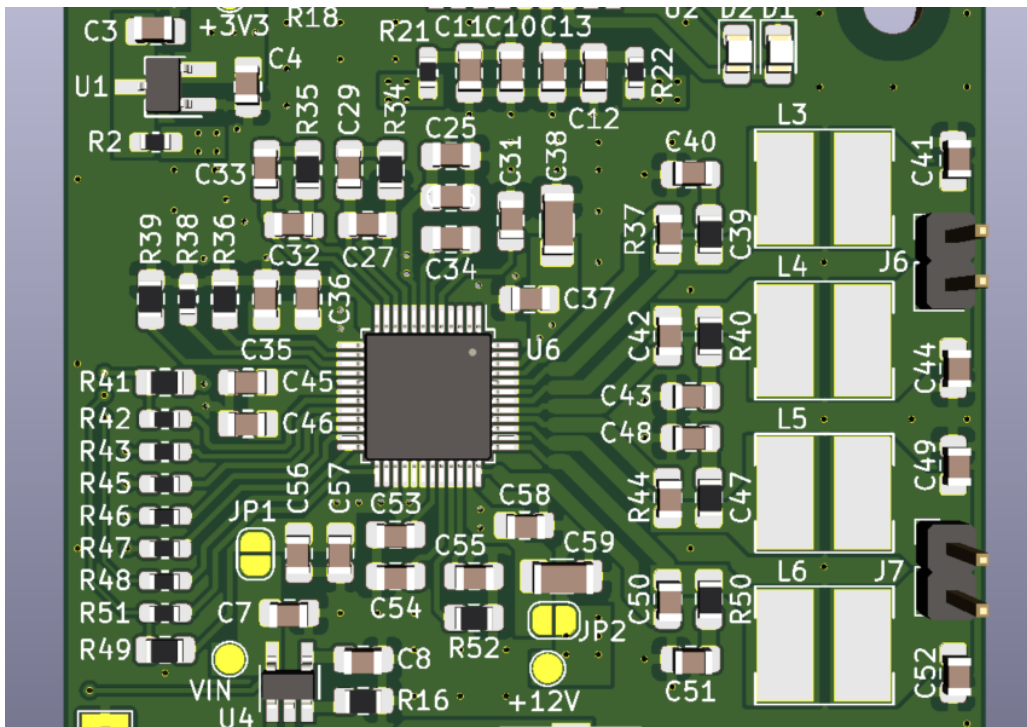


Abbildung 1: Beispiel einer bestückten Leiterplatte.

welche relevante Informationen über die zu bestückenden Bauteile gespeichert hat. Zu diesen Informationen gehören unter anderem die Anzahl, die Bezeichnung auf der Leiterplatte, der Typ/Wert des Bauteils, die Bauform, der Lieferant des Bauteils und die zugehörige Bestellnummer sowie der Stückpreis. Die Informationen für die Stückliste werden normalerweise direkt aus der PCB Design Software exportiert und in einer Textdatei gespeichert. Diese Textdatei ist häufig ein CSV Format (CSV = Comma-separated values), wobei das Trennzeichen zwischen einzelnen Datenfeldern (Spalten) nicht zwingenderweise ein Komma, sondern auch ein anderes Zeichen, wie zum Beispiel ein Semikolon sein kann. Der Inhalt eines solchen - mit Semikolon getrennten Datenfeldern - kann bspw. wie folgt aussehen¹:

```
Reference;Value;Description;Package;Part Number;Supplier;Supplier Part Number;Price
C1;1nF;Ceramic Capacitor;0805;0805N102J500CT;Digikey;1292-1591-1-ND;0.1
C2;10nF;Ceramic Capacitor;0805;CL21B103JBANNNC;Digikey;1276-1245-1-ND;0.1
C3;47nF;Ceramic Capacitor;0805;CL21B473KBCNNNC;Digikey;1276-1250-1-ND;0.1
C4;47nF;Ceramic Capacitor;0805;CL21B473KBCNNNC;Digikey;1276-1250-1-ND;0.1
C5;56pF;Ceramic Capacitor;0805;CL21C560JBANNNC;Digikey;1276-1833-1-ND;0.1
D1;green;LED;805;150080VS75000;Digikey;732-4986-1-ND;0.18
.
.
.
```

¹Der Übersichtlichkeit halber wird hier nur ein Ausschnitt der gesamten Stückliste gezeigt.

Die gesamte Stückliste in einer formatierten Tabelle:

Reference	Value	Description	Package	Part Number	Supplier	Supplier Part Number	Price
C1	1nF	Ceramic Capacitor	0805	0805N102J500CT	Digikey	1292-1591-1-ND	0.1
C2	10nF	Ceramic Capacitor	0805	CL21B103JBANNNC	Digikey	1276-1245-1-ND	0.1
C3	47nF	Ceramic Capacitor	0805	CL21B473KBCNNNC	Digikey	1276-1250-1-ND	0.1
C4	47nF	Ceramic Capacitor	0805	CL21B473KBCNNNC	Digikey	1276-1250-1-ND	0.1
C5	56pF	Ceramic Capacitor	0805	CL21C560JBANNNC	Digikey	1276-1833-1-ND	0.1
D1	green	LED	0805	150080VS75000	Digikey	732-4986-1-ND	0.18
D2	green	LED	0805	150080VS75000	Digikey	732-4986-1-ND	0.18
L1	4.7uH	Coil	nonstandard	7447789004	Digikey	732-1175-1-ND	2.34
L2	4.7uH	Coil	nonstandard	7447789004	Digikey	732-1175-1-ND	2.34
R1	10k	Chip Resistor	0805	RC0805FR-0710KL	Digikey	311-10.0KCRCT-ND	0.1
R2	10k	Chip Resistor	0805	RC0805FR-0710KL	Digikey	311-10.0KCRCT-ND	0.1
R3	10k	Chip Resistor	0805	RC0805FR-0710KL	Digikey	311-10.0KCRCT-ND	0.1
R4	100k	Chip Resistor	0805	RC0805FR-07100KL	Digikey	311-100KCRCT-ND	0.1
R5	100k	Chip Resistor	0805	RC0805FR-07100KL	Digikey	311-100KCRCT-ND	0.1
R6	100k	Chip Resistor	0603	RC0603FR-07100KL	Digikey	311-100KCRCT-ND	0.1
R7	680R	Chip Resistor	0805	CRGCQ0805F680R	Digikey	A129747CT-ND	0.1
U1	MCP1700	Linear Voltage Regulator	SOT-23-3	MCP1700T-3302E/TT	Digikey	MCP1700T3302ETTCT-ND	0.37
U2	BM64	Bluetooth Module	module	BM64SPKS1MC1-0001AA	Digikey	BM64SPKS1MC1-0001AA-ND	13.65
U3	MSP430G2955	Microcontroller	38-TSSOP	MSP430G2955IDA38R	Digikey	296-36686-1-ND	3.09
U4	TPS73033	Linear Voltage Regulator	SOT-23-3	TPS73033DBVT	Digikey	296-27057-1-ND	0.81

Bevor man die Leiterplatte bestücken kann, muss man natürlich die entsprechenden Bauteile aus dem Lager oder - falls die Bauteile nicht vorhanden sind - von einem Lieferanten beziehen. In der folgenden Testaufgabe sollen zwei Klassen **Storage** sowie **BOM** (Bill Of Materials) implementiert werden. Mit der Klasse **Storage** und ihren Methoden kann dabei das Bauteilelager verwaltet werden und mit Hilfe der Klasse **BOM** kann - auch auf Grundlage der Informationen aus dem Lager - eine Liste der zu Verfügung (oder eben nicht zu Verfügung) stehenden Bauteile erstellt werden.

2 Aufgabenstellung und Bewertungskriterien

☞ Implementieren Sie die Klassen **Storage** und **BOM**, wie unten im Detail beschrieben, und liefern Sie diese als einzelne Python-Datei der Dozierenden bis spätestens am 21. April 2020 um 12 Uhr per Email (smalacar@hsr.ch) ab.

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 gibt es total 25 Punkte.

Die unten angegebenen **Klassen-, Methoden-, Attribut-, Parameter- und Key-Namen müssen exakt übernommen** werden, sonst werden für den betroffenen Teil keine Punkte vergeben. Es wird empfohlen, den eigenen Code ausgiebig zu testen. Die Beispiele in den grauen Listing-Boxen eignen sich für einen ersten Vergleich.

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

3 Allgemeine Bewertungskriterien

- Nur die unten aufgelisteten Methoden dürfen *public* sein und die Klassen besitzen keine *public* Instanzvariablen. 1 P
- Die Klassen und deren Methoden besitzen aussagekräftige Docstrings. 1 P
- Python-Codestil entspricht den PEP8-Empfehlungen. 0.5 P Abzug pro Stilfehler². 2 P

²der Codestil wird mit der Flake8-Software geprüft, <http://flake8.pycqa.org/en/latest/>

4 Klasse Storage

Diese Klasse ist für die Bauteilverwaltung zuständig. Beim Lager handelt es sich um einen lokalen Ordner auf dem Computer, in welchem *.txt Files abgelegt sind. Jedes Bauteil (bzw. jede Part Number) erhält dabei sein eigenes <part_number>.txt File in diesem Ordner, in welchem die Anzahl der im Lager befindlichen Bauteile sowie zusätzliche Informationen über das Bauteil gespeichert werden. Als Filename soll die entsprechende Part Number `part_number` gewählt werden. Das folgende Beispiel zeigt den Inhalt eines solchen Files anhand des Bauteiles mit Part Number RC0805FR-07100KL:

```
count=100
description=Chip Resistor
value=100k
unit=0hm
package=0805
tolerance=0.01
power=0.125
supplier=Digikey
supplierpartnumber=311-330CRCT-ND
price=0.01560
currency=CHF
```

Die im File gespeicherten Informationen können von Bauteil zu Bauteil verschieden sein und auch die Reihenfolge muss nicht zwingend wie oben angegeben sein. Grundsätzlich gilt: Eine Information pro Zeile und jedem gegebenen Attributnamen (`count`, `description`, `value`, etc.) wird über ein Gleichheitszeichen genau ein Wert zugeordnet. Leerzeichen vor und nach dem Gleichheitszeichen sowie am Ende der Zeile sollen ignoriert werden. Sie finden einige Beispielfiles im beigelegten Ordner `storage`.

4.1 Methode `__init__(self, folder_path)`

(3 P)

- Die Methode liest aus allen <part_number>.txt Files (`encoding="utf-8"`) im angegebenen Ordner im Pfad `folder_path` alle Attribute in ein internes Dictionary `self._data` ein. Leere Zeilen werden ignoriert. (1 P)
Hinweis: Benutzen Sie die Funktion `glob()`³, um eine Liste der *.txt-Files zu erstellen.
- Der Parameter `folder_path` soll in einer internen Instanzvariablen `self._path` gespeichert werden. (1 P)
- Die Methode speichert alle Attribut-Informationen in einem internen Dictionary `self._data`, wobei der Key jeweils die Part Number – welche direkt vom Filenamen <part_number>.txt übernommen werden kann – und der zugehörige Wert ein Dictionary mit allen Attributnamen und Attributwerten aus dem jeweiligen <part_number>.txt File ist. Alle Werte sollen als `str` beibehalten werden. (1 P)

³<https://docs.python.org/3/library/glob.html>

Die Instanziierung eines **Storage**-Objektes kann wie folgt lauten:

```
>>> s = Storage("D:/Documents/storage")
```

Für das oben gezeigte File **RC0805FR-07100KL.txt** und ein weiteres File **TPS73033DBVT.txt** sieht dieses Dictionary wie folgt aus:

```
>>> print(s._data)
{"RC0805FR-07100KL": {
    "count": "100",
    "description": "Chip Resistor",
    "value": "100k",
    "unit": "Ohm",
    ...},
"TPS73033DBVT": {
    "count": "20",
    "value": "TPS73033",
    "package": "SOT-23-5"},
...}
```

4.2 Methode `create(self, part_number, attributes, overwrite=False)`

(3 P)

Diese Methode legt ein neues Bauteil im Bauteilelager an und nimmt dafür folgende Parameter entgegen:

<code>part_number</code>	Die Part Number des neuen Bauteiles. Mit diesem Parameter sowie der Instanzvariablen <code>self._path</code> kann der entsprechende Filepfad auf das neu zu erstellende <code><part_number>.txt</code> File erstellt werden.
<code>attributes</code>	Das Dictionary <code>attributes</code> beinhaltet alle relevanten Attributinformationen für dieses neue Bauteil und zwar in folgender Form: <code>{"count": 20, "value": "TPS73033", "package": "SOT-23-5", ...}</code> Im Dictionary <code>attributes</code> gegebene Zahlenwerte (z.Bsp. für <code>"count"</code> oder <code>"price"</code>) können neben <code>str</code> auch <code>int</code> oder <code>float</code> Objekte sein.
<code>overwrite</code>	Dieser boolesche Parameter gibt an, ob bereits bestehende Bauteilinformationen im File <code><part_number>.txt</code> sowie auch in den internen Daten <code>self._data</code> überschrieben werden sollen oder nicht. Der Defaultwert ist <code>False</code> : <code>overwrite=True</code> Die Informationen im File sowie in den internen Daten werden in jedem Fall überschrieben. <code>overwrite=False</code> Falls dieses Bauteil bereits existiert, wird nichts an den internen Daten <code>self._data</code> sowie dem File <code><part_number>.txt</code> geändert.

- Die Methode erstellt ein Textfile mit dem Dateinamen `<part_number>.txt` im Ordner, welcher mit `self._path` spezifiziert wurde. Alle im Dictionary `attributes` gegebenen Attribute werden als Attribut-Wert-Paar zeilenweise in das Textfile geschrieben. Der Attri-

1 P

butname soll mit einem Gleichheitszeichen vom zugehörigen Wert getrennt werden (siehe File-Beispiel im Abschnitt 4).

```
# create new component
>>> s.create("150080VS7500",
             {"count": 25,
              "value": "green",
              "price": 0.18})

True
```

- Das neue Bauteil mit all seinen gegebenen Attributinformationen soll ebenfalls ins interne Dictionary `self._data` übernommen werden, wobei alle Werte zu `str` konvertiert werden sollen.

1 P

```
# show new component
>>> print(s._data)
{"150080VS7500": {"count": "25",
                  "value": "green",
                  "price": "0.18"},
 ...,
 "TPS73033DBVT": {"count": "20",
                  "value": "TPS73033",
                  "package": "SOT-23-5"}}
}
```

- Die Methode berücksichtigt den Parameter `overwrite`. Falls weder die internen Daten `self._data` noch das File `<part_number>.txt` geändert oder neu erstellt worden sind, gibt die Methode `False` zurück, sonst `True`.

1 P

```
# try to create component
>>> s.create("TPS73033DBVT",
             {"count": 20,
              "price": 0.81},
             overwrite=False)

False
```

4.3 Methode `update(self, part_number, attributes)`

(3 P)

Diese Methode ergänzt oder ändert bereits bestehende Informationen über ein Bauteil im Lager. Die Methode nimmt dafür folgende Parameter entgegen:

<code>part_number</code>	Der Parameter gibt die Part Number des zu ändernden Bauteils an.
<code>attributes</code>	Das Dictionary <code>attributes</code> beinhaltet alle relevanten Attributinformationen, des zu ändernden Bauteils und zwar in folgender Form: <code>{"price": 0.5, "package": "SOT-23-5", ...}</code> Im Dictionary <code>attributes</code> gegebene Zahlenwerte (z.Bsp. für <code>"price"</code>) können neben <code>str</code> auch <code>int</code> oder <code>float</code> Objekte sein.

- Falls die angegebene `part_number` nicht existiert, wird eine `KeyError`-Exception mit dem Hinweis `"<part_number> is not present"` geworfen, wobei `<part_number>` die entsprechende Part Number, welche nicht existiert, angeben soll.

1 P

- Die Methode überschreibt oder ergänzt die neuen Attributinformationen im entsprechenden File `<part_number>.txt` im Ordner, welcher mit `self._path` spezifiziert wird. Das Attribut `"count"`, falls im Dictionary `attributes` vorhanden, muss ignoriert werden und darf nicht im File überschrieben werden. Alle anderen Attribute im Dictionary `attributes` werden als Attribut-Wert-Paar zeilenweise in das Textfile geschrieben. Der Attributname soll mit einem Gleichheitszeichen vom zugehörigen Wert getrennt sein. Tipp: Verwenden Sie dazu die Methode `create()`.
- Die gleichen Attribute, wie oben beschrieben, sollen ebenfalls ins interne Dictionary `self._data` übernommen werden, wobei alle Werte zu `str` konvertiert werden sollen.

1 P

1 P

Der Aufruf der Methode kann beispielsweise wie folgt aussehen:

```
# update component
>>> print("Before Update:", s._data["TPS73033DBVT"], sep="\n")
Before Update:
{"count": "20", "value": "TPS73033", "package": "SOT-23-5"}
>>> s.update("TPS73033DBVT", {"price": 0.81})
>>> print("After Update:", s._data["TPS73033DBVT"], sep="\n")
After Update:
{"count": "20", "value": "TPS73033", "package": "SOT-23-5", "price":
  "0.81"}
```

4.4 Methode `search(self, part_number=None, attributes={})` (3 P)

Diese Methode durchsucht das Lager nach Bauteilinformationen und zwar entweder auf Grund einer gegebenen Part Number oder eines gegebenen Dictionary mit den gesuchten Bauteilattributen:

<code>part_number</code>	Der Parameter gibt die Part Number des gesuchten Bauteiles an. Defaultwert ist <code>None</code> .
<code>attributes</code>	Das Dictionary <code>attributes</code> gibt die gesuchten Attributinformationen an und zwar bspw. in folgender Form: <code>{"value": "100k", "package": "0603"}</code> . Der Defaultwert ist ein leeres Dictionary. Im Dictionary <code>attributes</code> gegebene Zahlenwerte (z.Bsp. für <code>"price"</code>) können neben <code>str</code> auch <code>int</code> oder <code>float</code> Objekte sein.

- Wurde die `part_number` angegeben (nicht `None`) und existiert sie in den Daten, so gibt die Methode ein Dictionary mit allen Attributen des gegebenen Bauteils zurück:

1 P

```
# search part by given part number
>>> print(s.search(part_number="TPS73033DBVT"))
{"count": "20", "value": "TPS73033", "package": "SOT-23-5",
  "price": "0.81"}
```

- Hat `part_number` den Wert `None` und ist das Dictionary `attributes` nicht leer, so gibt die Methode alle zu den gegebenen Attributen in `attributes` passenden Bauteile als Dictionary zurück, sofern diese vorhanden sind, ansonsten soll `None` retourniert werden. Die gegebenen Attribute sollen bei der Suche UND-verknüpft werden:

1 P

```
# search parts by given attributes
>>> print(s.search(attributes={"value": "100k", "package":
    "0603"}))
{"RC0603FR-07100KL": {"count": "50", "value": "100k", "unit":
    "Ohm", "package": "0603", "description": "Chip Resistor",
    "supplier": "Digikey", "supplierpartnumber":
    "311-100KHRCT-ND", "price": "0.009", "currency": "CHF"}}
>>> print(s.search(attributes={"value": "999k", "package":
    "0603"}))
None
```

- Hat `part_number` den Wert `None` und ist `attributes` leer, gibt die Methode alle im Lager befindlichen Bauteile und deren Attribute zurück:

1 P

```
# search parts
>>> print(s.search())
{"150080VS7500": {"count": "25", "value": "green", "price":
    "0.18"}, "RC0603FR-07100KL": {"count": "50", "value": "100k",
    "unit": "Ohm", "package": "0603", "description": "Chip
    Resistor", "supplier": "Digikey", "supplierpartnumber":
    "311-100KHRCT-ND", "price": "0.009", "currency": "CHF"},
    "RC0805FR-07100KL": {"count": "100", "description": "Chip
    Resistor", "value": "100k", "unit": "Ohm", "package": "0805",
    "tolerance": "0.01", "power": "0.125", "supplier": "Digikey",
    "supplierpartnumber": "311-330CRCT-ND", "price": "0.01560",
    "currency": "CHF"}, "TPS73033DBVT": {"count": "20", "value":
    "TPS73033", "package": "SOT-23-5", "price": "0.81"}}
```

4.5 Methode `take(self, parts)`

(2 P)

Diese Methode entnimmt dem Lager Bauteile und ändert die zugehörigen `count`-Werte. Die Methode nimmt dafür folgende Parameter entgegen:

parts Das Dictionary `parts` beinhaltet alle Bauteile, welche aus dem Lager entnommen werden sollen mit den dazu gehörenden Mengenangaben als `int`-Werte und zwar in folgender Form:

```
{"TPS73033DBVT": 20, "RC0805FR-07100KL": 120, ...}
```

- Falls ein angefordertes Bauteil im Lager in genügender Menge existiert, wird die angeforderte Menge vom `count`-Wert subtrahiert. Falls das Bauteil existiert, aber in ungenügender Menge vorhanden ist, wird der `count`-Wert vom Bauteil auf 0 gesetzt. Die `count`-Werte werden sowohl im internen Dictionary `self._data` als auch in den entsprechenden Files `<part_number>.txt` aktualisiert, wobei die Werte zu `str` konvertiert werden sollen.
- Falls nicht alle Bauteile in der angeforderten Menge (oder gar nicht) vorhanden sind, gibt die Methode ein Dictionary zurück, welches die fehlenden Mengen (als negative Zahlen) der jeweiligen Bauteilen enthält. Hat es im Lager bspw. 20 Bauteile, aber das Dictionary `parts` gibt bei diesem Bauteiltyp die Menge 25 an, so wird von diesem Bauteiltyp der Wert -5 zurück gegeben. Sind alle Bauteile in genügender Menge vorhanden, wird `None` zurückgegeben.

1 P

1 P

```
>>> print(s.take({"TPS73033DBVT": 25, "HSMC-C190": 13}))
{"TPS73033DBVT": -5, "HSMC-C190": -13}

>>> print(s.take({"150080VS7500": 5}))
None
```

4.6 Methode `add(self, parts)`

(2 P)

Diese Methode füllt Bauteile ins Lager und ändert die zu den Bauteilen zugehörigen `count`-Werte. Die Methode nimmt dafür folgende Parameter entgegen:

parts Das Dictionary `parts` beinhaltet alle Bauteile, welche ins Lager gefüllt werden sollen mit den dazu gehörenden Mengenangaben als `int`-Werte und zwar in folgender Form:

```
{"TPS73033DBVT": 20, "RC0805FR-07100KL": 120, ...}
```

- Die Methode addiert die im Dictionary `parts` gegebenen Mengen zum `count`-Wert des zugehörigen Bauteils, aber nur falls das jeweilige Bauteil bereits existiert. Die `count`-Werte werden sowohl im internen Dictionary `self._data` als auch in den entsprechenden Files `<part_number>.txt` aktualisiert, wobei die Werte zu `str` konvertiert werden sollen. Tipp: Verwenden Sie dazu die Methode `create()`. (1 P)
- Sollten sich Bauteile im `parts` Dictionary befinden, welche im Lager nicht existieren, so werden diese (und nur diese) in Form eines Dictionaries wieder zurückgegeben. Konnten alle Bauteile im gegebenen Dictionary `parts` eingefüllt werden, so wird `None` zurückgegeben. (1 P)

```
>>> print(s.add({"150080VS7500": 30})) # existent
None
```

```
>>> print(s.add({"HSMC-C190": 13})) # non-existent
{"HSMC-C190": 13}
```

5 Klasse BOM

5.1 Methode `__init__(self, csv_file, delimiter=";")`

(2 P)

csv_file Gibt den Pfad zur Stückliste, die als CSV-File gespeichert ist, an. Die erste Zeile im CSV-File beinhaltet die Überschriften der Spalten, z.B.:
Reference;Value;Description;Package;Part Number;Supplier;Supplier Part Number;Price
C1;1nF;Ceramic Capacitor;0805;0805N102J500CT;Digikey;1292-1591-1-ND;0.1
...

delimiter Definiert mit welchem Zeichen die Spaltenwerte im File getrennt sind.

- Die Methode liest die im Pfad `csv_file` angegebene Stückliste (`encoding="utf-8"`) ein und speichert die Daten im internen Dictionary `self._parts`. Dabei sollen nur die Spalten mit "Reference" und "Part Number" berücksichtigt werden. Die Reihenfolge der Spalten (1 P)

im File kann beliebig sein. Leere Zeilen werden ignoriert. Die Instanzvariable `self._parts` ist ein Dictionary von Dictionaries und ist wie folgt aufgebaut⁴:

```
>>> b = BOM("stueckliste.csv")
>>> print(b._parts)
{"0805N102J500CT": {"reference": ["C1"]},
 "CL21B103JBANNNC": {"reference": ["C2"]},
 "CL21B473KBCNNNC": {"reference": ["C3", "C4"]},
 ...}
```

Beim Key des Dictionary `self._parts` handelt es sich jeweils um die Part Number eines Bauteils. Der zugehörige Wert ist wiederum ein weiteres Dictionary, dessen Key "reference" heisst und der zugehörige Wert ist jeweils eine Liste, welche alle zur Part Number gehörenden Referenzen beinhaltet.

- Wenn mehrere Bauteile dieselbe Part Number haben (bspw. am obigen Beispiel die Bauteile "C3" und "C4"), sollen diese Referenzen in alphabetischer Reihenfolge sortiert in der Liste abgespeichert werden.

1 P

5.2 Methode `availability(self, storage, units=1, output_text=False)`

(3 P)

Diese Methode erstellt eine Übersicht aller verfügbaren Bauteile in Form eines Dictionaries anhand der intern abgespeicherten Stücklistendaten `self._parts` und eines Objektes `storage` der Klasse `Storage`. Die Methode nimmt weiterhin folgende Parameter entgegen:

<code>storage</code>	Ein Objekt der Klasse <code>storage</code> .
<code>units</code>	Die Anzahl PCBs, welche bestückt werden sollen. Der Parameter ist vom Datentyp <code>int</code> . Defaultwert ist 1.
<code>output_text</code>	Diese Boolesche Variable gibt an, ob zusätzlich ein formatierter Text der Übersicht aller verfügbaren Bauteile zurück gegeben werden soll. Defaultwert ist <code>False</code> .

- Die Methode liefert anhand der Daten im Dictionary `self._parts` und dem Objekt `storage` ein Dictionary von Dictionaries in folgender Form zurück:

1 P

```
{part_number: {
    "available": available,
    "required": required,
    "missing": missing,
  }, ...}
```

Für jede `part_number` in `self._parts` wird im oben gezeigten Dictionary ein Dictionary erstellt, wobei die Schlüssel-Wert-Paare wie folgt zu interpretieren sind:

"available": `available`

Die im Lager befindliche Anzahl Bauteile (anhand des `count`-Attributs) des entsprechenden Typs `part_number`.

⁴Als Beispiel die ersten drei Part Number der angegebenen Stückliste.

"required": required

Die benötigte Anzahl dieses Bauteiltyps in der Stückliste `self._parts` multipliziert mit der Anzahl gewünschter PCBs, die mit dem Parameter `units` angegeben wird. Der Wert von `units` muss > 0 , ansonsten wird mit dem default-Wert `units = 1` gerechnet.

"missing": missing

Die fehlende Anzahl Bauteile des entsprechenden Typs `part_number`, d.h. die Differenz aus "required" und "available".

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

- Sollte eine entsprechende `part_number` nicht im Lager aufzufinden sein (das File `<part_number>.txt` im Ordner `storage._path` fehlt), wird im zurück gegebenen Dictionary beim Schlüsselwert der entsprechenden `part_number` der Wert `missing` auf den Wert `required` gesetzt:

1 P

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

Der Aufruf der Methode kann beispielsweise wie folgt aussehen:

```
# inventory
>>> print(b.availability(storage=s, units=10))
{"TPS73033DBVT": {"required": 10, "available": 20, "missing": 0},
 "0805N102J500CT": {"required": 10, "available": 0, "missing": 10},
 "CL21B103JBANNNC": {"required": 10, "available": 0, "missing": 10},
 "CL21B105KAFNNNE": {"required": 40, "available": 0, "missing": 40},
 "150080VS75000": {"required": 20, "available": 0, "missing": 20},
 "7447789004": {"required": 20, "available": 0, "missing": 20},
 "1255AY-150M=P3": {"required": 40, "available": 0, "missing": 40},
 "RC0805FR-0710KL": {"required": 30, "available": 0, "missing": 30},
 "RC0805FR-07100KL": {"required": 20, "available": 100, "missing": 0},
 "ESR10EZPJ270": {"required": 20, "available": 0, "missing": 20},
 "MCP1700T-3302E/TT": {"required": 10, "available": 0, "missing": 10},
 "BM64SPKS1MC1-0001AA": {"required": 10, "available": 0, "missing": 10},
 "MSP430G2955IDA38R": {"required": 10, "available": 0, "missing": 10}}
```

- Wird der Parameter `output_text=True` gesetzt, dann wird zusätzlich zum obigen Dictionary auch ein formatierter String zurückgegeben. Dabei sollen vier Spalten ausgegeben werden: Part Number, available, required, missing. Die Zeilen sollen anhand der Part Number in alphabetischer Reihenfolge sortiert ausgegeben werden. Die Spaltenbreiten sollen exakt so gross sein, wie das längste Element in der jeweiligen Spalte. Die erste Spalte ist linksbündig, die anderen drei Spalten sind rechtsbündig anzuordnen. Zwischen den Spalten soll ein Leerzeichen platziert werden.

1 P

Ein Beispiel sieht wie folgt aus:

```
>>> d, txt = b.availability(storage=s, units=10, output_text=True)
>>> print(txt)
Part Number          available required missing
=====
0805N102J500CT         0         10         10
1255AY-150M=P3         0         40         40
150080VS75000          0         20         20
7447789004             0         20         20
BM64SPKS1MC1-0001AA    0         10         10
```

CL21B103JBANNNC	0	10	10
CL21B105KAFNNNE	0	40	40
ESR10EZPJ270	0	20	20
MCP1700T-3302E/TT	0	10	10
MSP430G2955IDA38R	0	10	10
RC0805FR-07100KL	100	20	0
RC0805FR-0710KL	0	30	30
TPS73033DBVT	20	10	0