Klausur Einführung in die Programmierung I

Bjoern Stuetz (bstuetz@lehre.dhbw-stuttgart.de)

2021-12-17

Klausur

Übersicht

- Klausurdauer: **120 Minuten**
- Mögliche Gesamtpunktzahl: **120 Punkte**
- **Keine** Hilfsmittel (siehe *Informationen zur Verwendung von Python*)
- Verwenden Sie die ausgeteilten **Aufgabenblätter**
- Sollte der Platz nicht ausreichen, oder sollten Sie Aufgaben auf der Rückseite der Klausur oder auf Extrablättern lösen, vermerken Sie dies bitte auf dem Aufgabenblatt bei der entsprechenden Aufgabe.

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte	Kommentar
A	1	05	[ ]	
A	2	05	[ ]	
B	1	10	[ ]	
B	2	10	[ ]	
B	3	10	[ ]	
B	4	10	[ ]	
C	1	30	[ ]	
C	2	40	[ ]	
–	–	**Summe**	**Erreicht**	–
–	–	**120**	**[ ]**	–

Tabelle 1: Aufgabenteile und Punkte

Falls Ihnen nicht immer sofort die richtige Lösung einfällt, beschreiben Sie bitte Ihren Lösungsansatz kurz und knapp mit einer Schema-Zeichnung, in Pseudocode oder direkt in Worten.

Viel Erfolg!

Informationen zur 'Programmierung auf Papier'

Bitte nehmen Sie sich kurz Zeit für diesen Abschnitt. Dieser wurde auch bereits in der Vorlesung vorgestellt. Es sind alle Hilfsmittel erlaubt. Programmieren Sie in dieser Klausur konzeptionell (d.h. soweit wie möglich vollständig aber sinnvoll und zielorientiert vereinfacht). Vermeiden Sie unnötige Teile, die nicht ausdrücklich gefragt wurden. Die Syntax sollte größtenteils korrekt sein; es ist allerdings wichtiger Ihre Idee auszudrücken, als vollständig übersetzbaren (also compilierbaren) Code zu schreiben.

Vereinfachte Darstellung

Folgende Dinge können vereinfacht dargestellt werden (außer es wird ausdrücklich in der Fragestellung auf die reguläre Darstellung verwiesen):

- Ausgabe von Text muss nicht formatiert werden.
- Input von Text wird nie über die Konsole verwendet, Input-Variablen werden vorab zugewiesen.
- `__str__()` sind immer vorhanden, auch wenn diese nicht von Ihnen ausprogrammiert wurden.
- Module sind vorhanden und automatisch importiert.
- Standard-Konstruktoren, also die `__init__()` Methode, sind immer vorhanden, auch wenn diese nicht von Ihnen ausprogrammiert wurden.

Diese Punkte müssen entsprechend **nicht** ausprogrammiert bzw. voll ausgeschrieben werden.

Programmierstil

Bitte beachten Sie folgende Hinweise zum Programmierstil: Beachten Sie die Regeln und Konventionen der Programmiersprache. Programmieren Sie auf **deutsch** oder **englisch** (*engl.*); falls Ihnen nicht das richtige Wort in englisch einfällt, nutzen Sie einfach ein **deutsch**es Wort. Bitte denken Sie an die notwendigen Konventionen für Python (z.B. Einrückung, Variablen, Namen). Bitte rücken Sie den Programmcode so weit wie möglich ein.

Bitte schreiben Sie leserlich und verständlich.

Informationen zur Verwendung von Python

- Zuweisungen: Zuweisung mit `=`, z.B. `a = 1`, Vergleich mit `==`, Inkrement/Dekrement mit `+=`, `-=`
- Rechnen: Addition `+`, Subtraktion `-`, Multiplikation `*`, Division `/`, Ganzahldivision `//`, Potenzierung `**`, Restwert (Modulo) `%`
- Datentypen: String: 'String' "String" `str()`, Integer: 5 `int()`, Float: (Gleitkomma): 4.3 `float()`, Boolean `True` / `False` `bool()`
- Datenstrukturen: Listen `[]`, Tupel `()` und Dictionaries `{ }`
- String-Operationen: `*`, `'String' * 2`, `+`, `in`, `'S' in 'String'`
- String Auswahl: `'String'[3]`, `'String'[1:2]`

- String Methoden: hochstellen, `upper()`, tiefstellen, `lower()`, Zeichen zählen `count('c')`, Zeichen ersetzen `replace('a', 'b')`, Leerzeichen (*whitespace*) entfernen `strip()`
- Listen: `list = ['a', 'b', c, d]`
- Listen Auswahl, Zerteilen, Kopieren: Index 1, `list[1]`, Index n-3, `list[-3]`, Index 1 und 2 `list[1:3]`, nach Index 1, `list[1:]`, vor Index 3, `list[:3]`, Kopie, `list2 = list1[:]`
- Listen von Listen: `list = [[a, b], [c, d]]`, `list[1][0]`
- Listenoperationen: `+`, `*`, `>`
- Listenmethoden: Index `index()`, Anzahl `count()`, Anhängen `append(a)`, Einfügen an Position `insert(0, i)`, Entfernen `remove(r)`, `del(list[0:1])`, `pop(-1)`, Erweitern `extend(list2)`, Umkehren `reverse()`, Sortieren `sort()`
- Import: `import math`, `import math as m`, `from math import pi`
- Fallunterscheidung: `if`, `elif`, `else`
- Schleifen: `for`, `while`
- Funktionen: `def function1(arg1, arg2 = 'default')`, `return`, `function1('1')`
- Klassen: `class Class1(object)`, `def __init__(self)`, `def function1(self, arg1, arg2 = 'default')` `return`, `class1 = Class1()`, `class1.function1('1')`
- Ausnahmen: `try`, `except`, `else`, `finally`
- Keine Operation: `pass`
- Datentyp ermitteln `type()`, prüfen: `isinstance()`
- Ausgabe auf der Kommandozeile: `print()`, Eingabe: `input()`

Aufgabenteil A - Grundlagen

Aufgabe A1 - Hello World und Textausgabe

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
A	1	05	[ ]

Problem:

Erstellen Sie ein Python-Programm in der Datei `hello.py`, als Textausgabeprogramm,

- das die Zahlen `17`, `12` und `2021` (alle `integer`) durch Punkte (`.` als `string`) getrennt, also `17.12.2021`,

auf der Kommandozeile ausgibt.

Lösung:

Programmieren Sie bitte:

- (A1.1) Python-Programm `main.py`

```
# hello.py

# ausgabe 17.12.2021 mit zahlen und strings

```

Was ist die Ausgabe, wenn Sie das Ergebnis mit `print()` auf der Kommandozeile ausgeben?

`$ python3 hello.py`:

```
# (A1.2) ausgabe `17`, `12` und `2021` als `17.12.2021`


```

Aufgabe A2 - Datentypen und Typisierung

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
A	2	05	[ ]

Problem:

Erstellen Sie ein Python-Programm in der Datei `convert.py`, als Umrechenprogramm,

- das die Gleitzahl `1.7` (`float`)
- zu einer Ganzzahl (`int`),
- und dann wieder zu einer Gleitzahl (`float`)

ohne Rundungsregeln oder Rechenregeln umwandelt.

Lösung:

Programmieren Sie bitte:

- (A2.1) Python-Programm `convert.py` mit dem angegebenen Eingabewert:

```
# convert.py

# input
input = 1.7

# (A2.2) float zu int




# (A2.3) int zu float


```

Was ist die Ausgabe, wenn Sie die Zwischenergebnisse und das Ergebnis mit `print()` auf der Kommandozeile ausgeben?

`$ python3 convert.py`:

```
# (A2.2) ausgabe float zu int


```

```
5  # (A2.3) ausgabe int zu float
6
7
```

Aufgabenteil B - Weiterführende Themen

Aufgabe B1 - Einfaches Rechnen und Datentypen

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
B	1	10	[ ]

Problem:

Erstellen Sie ein Python-Programm in der Datei `calc.py`, als Rechenprogramm,

- die Gleitzahl `2.0` (`float`)
- mit der Ganzzahl `2` (`int`) multipliziert,

und das Ergebnis

- als Ganzzahldivision durch `2` (`int`) ohne Rundungsregeln oder Rechenregeln teilt,

und dann

- `1.5` (`float`) addiert

Lösung:

Programmieren Sie bitte:

- (B1.1) Python-Programm `calc.py` und **ergänzen** Sie die Eingabewerte:

```
# calc.py

# input

      =

      =


# (B1.2) multiplikation




# (B1.2) ganzzahl-division

```

```
18
19
20 # (B1.3) addition
21
22
23
```

Was ist die Ausgabe, wenn Sie die Zwischenergebnisse und das Ergebnis mit `print()` auf der Kommandozeile ausgeben?

`$ python3 calc.py`:

```
1 # (B1.4.1) ausgabe multiplikation
2
3
4 # (B1.4.2) ausgabe ganzzahl-division
5
6
7 # (B1.4.3) ausgabe addition
8
```

Von welchem Datentyp ist das Ergebnis? Warum ist das Ergebnis vom genannten Datentyp?

```
1 # (B1.5) datentyp
2
3 #
4 #
5
6 # (B1.6) warum dieser datentyp
7
8 #
9 #
```

Aufgabe B2 - Textausgabe in Funktionen

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
B	2	10	[ ]

Problem:

Erstellen Sie ein Python-Programm in der Datei `print.py`, als Textausgabeprogramm,

- mit einer Funktion `print_text()`,
  - die einen (1) Parameter mit Namen `text` hat,
  - der Parameter `text` hat als Standardwert (*default value*) eine leere Zeichenkette `""` (`string`).

Innerhalb der Funktion

- wird der lokalen (Zwischen-)variable `output` der Text `"Hallo"` (`string`) verkettet mit dem Parameter `text` zugewiesen,

und das Ergebnis auf der Kommandozeile ausgegeben.

Lösung:

Programmieren Sie bitte:

- (B2.1) Python-Programm `print.py` mit dem angegebenen Eingabewert:

```
# print.py

# input
input = "Du!"

# (B2.2) funktion mit parameter und standard-wert, textausgabe
```

Wie rufen Sie die Funktion auf? Nutzen Sie als Parameter die Variable `input`.

```
# (B2.3) funktions-aufruf
```

Was ist die Ausgabe auf der Kommandozeile?

`$ python3 print.py`:

```
# (B2.4) ausgabe funktions-aufruf


```

Was ist eine Funktion in Python? Wie gebe ich Werte aus einer Funktion zurück? Warum nutzen wir Funktionen und was sind Vorteile von Funktionen?

```
# (B2.5) was ist eine funktion, wie gebe ich werte zurück

#
#
#

# (B2.6) warum nutzen wir funktionen, was sind vorteile von funktionen

#
#
#
```

Aufgabe B3 - Rechnen mit Funktionen und Verzweigungen

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
B	3	10	[ ]

Problem:

Erstellen Sie ein Python-Programm in der Datei `divide.py`, als Rechenprogramm,

- mit einer Funktion `divide()`,
  - die zwei (2) Parameter mit Namen `dividend` und `divisor` hat, aber **ohne** Standardwerte (*default values*)

Innerhalb der Funktion

- wird als Ergebnis der erste Parameter `dividend` durch den zweiten Parameter `divisor` geteilt,
- falls der zweite Parameter `divisor` gleich Null (`0`) ist, ist Null (`0`) das Ergebnis,

und das Ergebnis ohne Textausgabe zurückgegeben.

Nach dem Funktionsaufruf wird das Ergebnis auf der Kommandozeile ausgegeben.

Lösung:

Programmieren Sie bitte:

- (B3.1) Python-Programm `divide.py` mit den angegebenen Eingabewerten:

```
# divide.py

# input
input1 = 2
input2 = 1

# (B3.2) funktion mit zwei parameter und berechnung
```

Wie rufen Sie die Funktion auf? Nutzen Sie als Parameter die Variablen `input1` und `input2`.

```
# (B3.3) funktions-aufruf


```

Was ist die Ausgabe auf der Kommandozeile?

`$ python3 divide.py`:

```
# (B3.4) ausgabe funktions-aufruf


```

Was ist eine Verzweigung in Python? Warum nutzen wir Verzweigungen?

```
# (B3.5) was ist eine verzweigung

#
#
#

# (B3.6) warum nutzen wir verzweigungen

#
#
#
```

Aufgabe B4 - Textausgabe in einer Schleife mit Rechnen und Fallunterscheidung

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
B	4	10	[ ]

Problem:

Erstellen Sie ein Python-Programm in der Datei `countdown.py`, als Countdown-Programm bis Silvester,

- mit eine Schleife, die von eins (`1`) bis `31` zählt,
- und mit einer Funktion `countdown()`,
  - die einen (1) Parameter mit Namen `day` (Tag) hat,

Die Funktion wird pro Schleifendurchlauf mit der Laufvariable bzw. dem Zähler des aktuellen Schleifendurchlaufs (z.B. `i`) aufgerufen.

Innerhalb der Funktion

- wird der Parameter `day` auf auf der Kommandozeile ausgegeben, dabei `"Tag:"` der Ausgabe vorangestellt,
- die verbleibenden Tage bis zu Silvester (**31**.12.) berechnet und auf der Kommandozeile ausgegeben, dabei `"Tage bis Silvester:"` der Ausgabe vorangestellt
- ab dem 24. Durchlauf (**24**.12.) wird **zusätzlich** der Text `"Hurra, Urlaub!"` auf der Kommandozeile ausgegeben

Eine Rückgabe ist in der Funktion nicht vorhanden.

Lösung:

Programmieren Sie bitte:

- (B4.1) Python-Programm `countdown.py` mit den angegebenen Eingabewerten:

```
# countdown.py

# iterator
i = 1

# (B4.2) schleife und funktions-aufruf
```

```
# (B4.3) funktion mit einem parameter und ausgabe
```

Was ist die Ausgabe auf der Kommandozeile? Bitte verwenden Sie als Beispiel **nur** für den ersten (1., `i = 1`) und 25. (`i = 25`) Durchlauf!

`$ python3 countdown.py`:

```
# (B3.4.1) ausgabe funktions-aufruf für i = 1


# (B3.4.2) ausgabe funktions-aufruf für i = 25
```

Was ist eine Schleife in Python? Nennen Sie zwei (2) Beispiele. Warum nutzen wir Schleife?

```
# (B3.5) was ist eine schleife, zwei beispiele

#
#
#
#
#
#
#
```

```
10
11  # (B3.6) warum nutzen wir schleifen
12
13  #
14  #
15  #
```

Aufgabenteil C - Fallbeispiele mit Klassen und Objekten

Aufgabe C1 - Fallbeispiel Test-Verwaltung

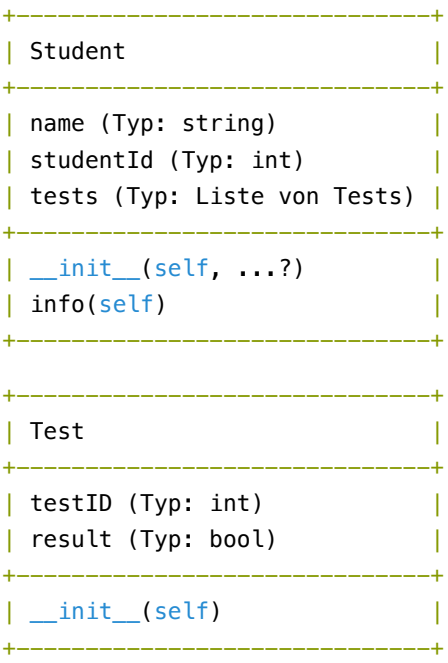
Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
C	1	30	[ ]

Hintergrund:

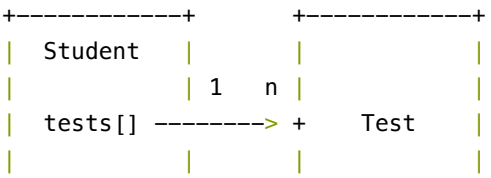
Sie werden von der DHBW gebeten, einen ersten Entwurf eines Programms für die Verwaltung von Tests für Studenten zu erstellen.

Für das Programm nutzen Sie zwei Klassen `Student` und `Test`. Einem Student können mehrere Tests zugewiesen werden, die jeweils den Zustand `True` oder `False` haben.

Klassendiagramme:



Übersicht:



```
|             |       |             |
+-------------+       +-----+-------+
```

Problem:

Erstellen Sie ein Python-Programm in der Datei `test.py`, als Verwaltungsprogramm für Corona-Tests.

Erstellen Sie die Klasse `Student` mit den Attributen

- `name` als `string`,
- `studentId` als `int`,
- `tests` als Liste von Objekten der Klasse `Test`,

und der Methode

- `info()`, die als Rückgabewert ohne Ausgabe `True` zurückgibt, sobald mindestens ein Test `True` ist

Getter und Setter sind für die Klasse `Student` nicht notwendig.

Erstellen Sie einen passenden Konstruktor (*initialiser*) für die Klasse `Student`, der eine Liste von `Test`s initialisiert,

- also eine leere Liste ohne Elemente anlegt.

Erstellen Sie die Klasse `Test` mit den Attributen

- `testID` als `int`,
- `result` als `boolean`

Getter und Setter, sowie ein Konstruktor sind für die Klasse `Test` nicht notwendig.

Legen Sie die vollständigen `Student`en-Objekte samt Test-Objekte in einer Liste anhand den vorgegebenen Eingabewerten an.

Lösung:

Programmieren Sie bitte:

- (C1.1) Python-Programm `test.py` mit den angegebenen Eingabewerten:

```
# test.py

# input
name1 = "Maxine Muster"
studentID1 = "1"
test1_1 = False
test1_2 = True

name2 = "Bert Beispiel"
studentID2 = "2"
```

```
test2 = False

# (C1.2) klasse student mit konstruktor und methoden

class Student:


    __init__(self,









# (C1.3) klasse test

class Test:
```

```
54 # (C1.4) anlage objekte
55
56
57
58
59 student1 =
60
61
62
63
64
65
66 student2 =
67
68
69
70
71
```

Welchen Wert und Datentyp hat die Variable `student1.tests[1].result` für die angegebenen Eingabewerte?

`$ python3 test.py`:

```
1 # (C1.4) wert und typ variable student1.tests[1].result
2
3 #
4 #
5 #
```

Was ist eine Klasse in Python? Was ist ein Objekt? Warum nutzen wir Objekte?

```
1 # (C1.5) was ist eine klasse, was ein objekt
2
3 #
4 #
5 #
6 #
7 #
8 #
9 #
10
11 # (C1.6) warum nutzen wir objekte
12
13 #
14 #
```

```
#
#
#
#
```

Aufgabe C2 - Fallbeispiel Digital Twin

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
C	2	40	[ ]

Hintergrund:

Sie werden von einem Automobilhersteller gebeten, einen ersten Entwurf eines Programms für die Verwaltung von Elektrofahrzeugen als digitale Zwillinge (*Digital Twins*) zu erstellen. Dieser digitale Zwilling bildet das Fahrzeug virtuell als Modell ab, und soll zuerst nur die Batterie und Ladestand abdecken.

Für das Elektrofahrzeug nutzen Sie die Klasse `Car`, mit den Methoden `charge` und `drive`. Ein Fahrzeug hat eine eine Batterie (Klasse `Battery`), die wiederum aus genau vier (4) Batteriezellen (Klasse `BatteryCell`) besteht, welche die eigentliche Ladung (*charge*) enthalten.

Klassendiagramme:

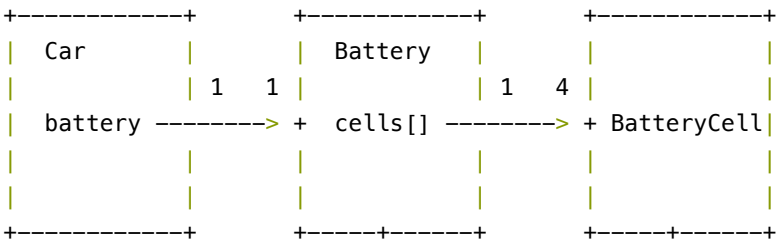
```
+------------------------------+
| Car                          |
+------------------------------+
| name (Typ: string)           |
| odometer (Typ: int)          |
| battery (Typ: Battery)       |
+------------------------------+
| __init__(self, ...?)         |
| charge(self, amount)         |
| drive(self, distance = 0)    |
| info()                       |
+------------------------------+

+------------------------------+
| Battery                      |
+------------------------------+
| cells (Typ: Liste von        |
|              BatteryCell)    |
+------------------------------+
| __init__(self, ...?)         |
+------------------------------+

+------------------------------+
| BatteryCell                  |
+------------------------------+
| charge = 100.0 (Typ: float)  |
```

```
+--------------------------------+
| __init__(self)                 |
+--------------------------------+
```

Übersicht:



Problem:

Erstellen Sie ein Python-Programm in der Datei `testdrive.py`, als Testprogramm für Digitale Zwillinge.

Erstellen Sie die Klasse `Car` mit den Attributen

- `name` als `string`,
- `odometer` (Kilometerzähler, Tachostand) als `int`, mit Standardwert (*default value*) `0`,
- `battery` als ein (1) Objekt der Klasse `Battery`,

und den Methoden

- `charge(self, amount)`, die Batterien werden aufgeladen, dabei kann die Gesamtladung (`charge`) minimal `0.0` und maximal `100.0` sein [`0` <= `charge` <= `100`],
- `drive(self, distance)`, das Fahrzeug fährt eine Distanz in Kilometer (`distance` in km),
  - die Distanz in Kilometer wird auf den Kilometerstand hinzugezählt
  - und das Fahrzeug verbraucht dabei pro Kilometer `4.0` Batterieladung,
- `info()`, die den Kilometerstand (`odometer`) und die Ladung (`charge` aller vier Batteriezellen) auf der Kommandozeile ausgibt

Getter und Setter sind für die Klasse `Car` nicht notwendig.

Erstellen Sie einen passenden Konstruktor (*initialiser*) für die Klasse `Car`, der die eine (1) Batterie initialisiert.

Erstellen Sie die Klasse `Battery` mit den Attributen

- `cells` als Liste von Objekten der Klasse `BatteryCell`

Getter und Setter sind für die Klasse `Battery` nicht notwendig.

Erstellen Sie einen passenden Konstruktor (*initialiser*) für die Klasse `Battery`, der die Liste von vier (4) `BatteryCell`s initialisiert.

Erstellen Sie die Klasse `BatteryCell` mit dem Attribut

- charge als `int`, mit Standardwert (*default value*) `100.0`

Getter und Setter, sowie ein Konstruktor sind für die Klasse `BatteryCell` nicht notwendig.

Legen Sie ein vollständiges `Car`-Objekte samt verbundener Objekte (`Battery`, Liste von `BatteryCell`s in `Battery`) an.

Lösung:

Programmieren Sie bitte:

- (C2.1) Python-Programm `testdrive.py` mit den angegebenen Eingabewerten:

```
# testdrive.py

# input
name = "Torsche Paycan"
distance1 = 10
distance2 = -1

# (C2.2) klasse car mit konstruktor und methoden

class Car:



    __init__(self,
```

```
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57 # (C2.3) klasse battery
58
59 class Battery:
60
61
62
63     __init__(self,
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
```

```
80
81
82
83  # (C2.4) klasse battery-cell
84
85  class BatteryCell:
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110 # (C2.5) anlage objekte
111
112 car = Car(
113
114
115
116
117
118
119
120
121
122
```

Was ist die Ausgabe von `car.info()` auf der Kommandozeile für folgende Beispiele?

`$ python3 testdrive.py`:

```
# testdrive.py

# (C2.6.1) ausgabe car.info() nach car.drive(distance1) mit 10









# (C2.6.2) ausgabe car.info() nach car.drive(distance2) mit -1
```

Welchen Wert und Datentyp hat die Variable `car.battery.cells[0].charge` für folgendes Beispiel nach `car.drive(distance1)`?

`$ python3 testdrive.py`:

```
# (C2.7) wert und typ variable car.battery.cells[0].charge nach car.drive(distance1) mit 10

#
#
#
```

Ende