University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatik I
Einführung in die Programmierung

Assessment-Prüfung HS18

Allgemeine Hinweise:

- Die maximale Punktzahl beträgt **90 Punkte**, erreichbar durch das Lösen aller Aufgaben.
- Die Bearbeitungszeit beträgt **90 Minuten**.
- Bitte überprüfen Sie, dass Sie alle 13 Seiten dieser Klausur erhalten haben.
- Benutzen Sie einen **schwarzen oder blauen, dokumentenechten Stift** für die Prüfung. Stifte mit **grüner oder roter** Farbe sowie **Bleistifte** sind **nicht gestattet**. Betroffene Antworten werden bei der Bewertung nicht berücksichtigt.
- Lassen Sie die Blätter dieser Klausur zusammengeheftet.
- Bitte schreiben Sie Ihren **Nachnamen** und Ihre **Matrikelnummer** auf die dafür vorgesehene Markierung am Ende **jeder Seite**.
- Sie dürfen eine **handgeschriebene Formelsammlung** verwenden (DIN-A5, beidseitig beschrieben), die klar mit Ihrem Namen gekennzeichnet ist.
- Studierende, deren Muttersprache nicht Deutsch ist, dürfen ein **Wörterbuch** verwenden.
- Die Verwendung von zusätzlichen Materialien **ist nicht gestattet**. Sollten Sie zu unfairen Mitteln greifen, nicht genehmigte Ressourcen verwenden oder von einem Kommilitonen abschreiben, wird die Klausur eingezogen und als nicht bestanden gewertet. Zusätzlich werden disziplinarische Massnahmen eingeleitet.
- Schreiben Sie Quelltexte in Python. Sie können dabei frei zwischen Version 2 und 3 und den entsprechenden Funktionen wählen. Es ist nicht erlaubt vordefinierte Funktionen zu verwenden, wenn deren Implementierung in der Aufgabenstellung gefordert ist.
- Sie finden am Ende der Prüfung eine Liste von hilfreichen Pythonfunktionen.
- Ändern Sie keine vorgegebene Methodensignaturen oder Variablennamen der Prüfung.
- **Durch Abgabe der Klausur bestätigen Sie:**
  - Ich habe diese Hinweise gelesen und verstanden.
  - Ich fühle mich körperlich und psychisch in der Lage an der Klausur teilzunehmen.
  - Der Arbeitsraum ist angemessenen und ich kann die Klausur störungsfrei bearbeiten.
- Während der Klausur auftretenden Störungen sind dem Aufsichtspersonal direkt zu melden.

Bitte füllen Sie die folgenden Felder in **grossen Druckbuchstaben** aus und schreiben Sie **deutlich**:

Nachname: ☐☐☐ _____ Vorname: _____

Matrikelnummer: ☐☐ - ☐☐☐ - ☐☐☐

Aufgabe 1	Aufgabe 2	Aufgabe 3	Aufgabe 4	Aufgabe 5	Aufgabe 6	Summe
20	10	16	20	14	10	90

Aufgabe 1: Allgemeine Fragen — 20 Punkte

Diese Aufgabe enthält einige kleine Pythonsnippets von denen jedes in der letzten Zeile einen Ausdruck enthält. Schreiben Sie den *Typ* und den *Wert* dieses Ausdrucks in die dafür vorgesehenen Felder. Lassen Sie das *Wert* Feld leer, wenn der Ausdruck keinen Wert hat. Sollte das Snippet einen Fehler enthalten, dann wählen Sie den *NoneType* als Typ und schreiben *error* als Wert.

Hinweis: Es genügt eine Nennung des einfachen Typnamens ohne Modul, z.B. *int* oder *integer*.

Hinweis: Die Snippets werden getrennt ausgeführt und haben keinerlei Seiteneffekte aufeinander.

Hinweis: Lesen Sie die Snippets sehr aufmerksam. Die Antwort ist nicht immer offensichtlich.

a) 2 Punkte

```
5/2 + 3
```

Typ: Wert:

b) 2 Punkte

```
b1 = "13579"
b2 = "02468"
b1[5:] + b2[-5]
```

Typ: Wert:

c) 2 Punkte

```
(lambda x: x%2 == 0)(2)
```

Typ: Wert:

d) 2 Punkte

```
d = [[[1, 1], [2, 2]], [[3, 3], [4, 4]]]
d[0][2]
```

Typ: Wert:

e) 2 Punkte

```
class X: pass
class Y(X): pass
class Z(Y): pass
if isinstance(Z(), X):
   e = 1
else:
   e = 2.3
e
```

Typ: Wert:

f) 2 Punkte

```
f = sorted({ 'a':1, 'b':2, 'c':3 }.items())
f[0]
```

Typ: Wert:

g) 2 Punkte

```
def g():
  return False
"x" if not g else {}
```

Typ: Wert:

h) 2 Punkte

```
def addition(arr):
  s = 0
  for el in arr:
    if el % 2 == 0:
      s += el
  return s
addition([1, 2, 3, 4])
```

Typ: Wert:

i) 2 Punkte

```
class Animal:
  def talk(self):
    return "Moo!"

class Dog(Animal):
  pass

dog = Dog()
dog.talk()
```

Typ: Wert:

j) 2 Punkte

```
class Employee:
   id = 0
   def __init__(self, name):
      self.name = name
      self.id = Employee.id
      Employee.id += 1

emp = Employee("Marc")
emp.id
```

Typ: Wert:

Aufgabe 2: Funktionen 10 Punkte

In dieser Aufgabe werden Sie einige Hilfsfunktionen schreiben. Das erforderliche Verhalten Ihrer Implementierung wird am Ende jeder Unteraufgabe mit Hilfe von `asserts` illustriert.

Hinweis: Sie müssen die Argumente nicht auf `None` überprüfen, sollen allerdings Randfälle überprüfen (z.B., negative Zahlen oder leere Strings).

a) Implementierung von `split` 5 Punkte

Schreiben Sie eine Funktion `split`, die einen gegebenen String bei jedem Leerzeichen teilt und die Wörter als Liste zurückgibt. Sie können davon ausgehen, dass der String nur Buchstaben und einzelne Leerzeichen enthält, keine Satz- oder Sonderzeichen.

```
def split(text):
```

```
assert split("") == []
assert split("aaa") == ["aaa"]
assert split("a bbb cc") == ["a", "bbb", "cc"]
```

b) Invertierter Index 5 Punkte

Schreiben Sie die Funktion `rev_idx`, die für eine Wortliste den *Invertierten Index* berechnet und zurückgibt: Ein `dictionary`, das jedes Wort als `key` und eine Liste als `value` enthält. Die Liste soll alle Indices speichern, an denen das Wort in der ursprünglichen Wortliste verwendet wird.

Hinweis: Die Funktion soll Gross- und Kleinschreibung nicht unterscheiden.

```
def rev_idx(words):
```

```
assert rev_idx([]) == {}
assert rev_idx(["a","b"]) == {"a": [0], "b": [1]}
assert rev_idx(["a","B","A","aa"]) == {"a": [0, 2], "aa": [3], "b": [1]}
```

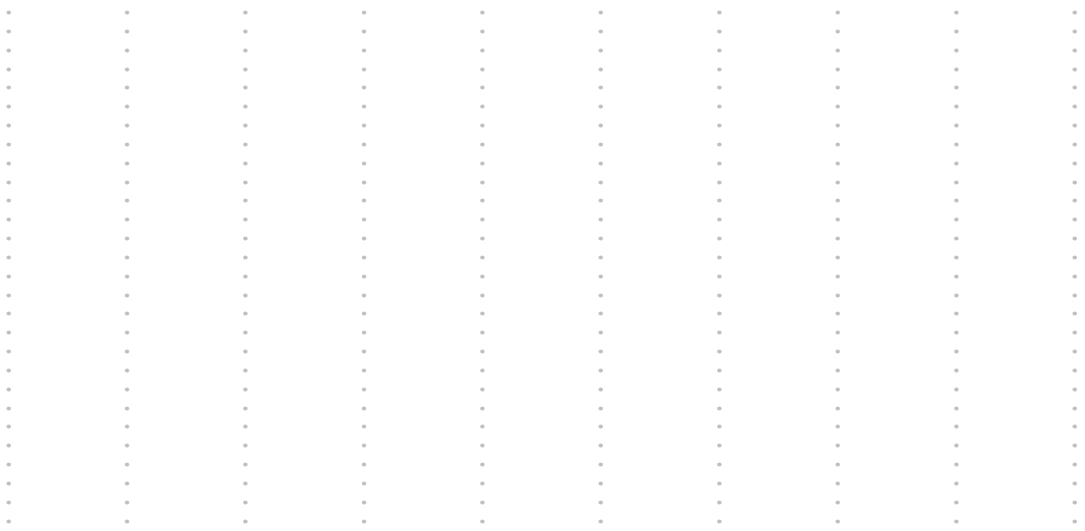
Aufgabe 3: Rekursion 16 Punkte

In dieser Aufgabe werden sie einige *rekursive* Hilfsfunktionen schreiben. Das erforderliche Verhalten Ihrer Implementierung wird am Ende jeder Unteraufgabe mit Hilfe von `asserts` illustriert.

a) Produkt zweier Zahlen 8 Punkte

Implementieren Sie die Funktion `prod`, welche zwei Zahlen *rekursiv* multipliziert. Sie können annehmen, dass `x` und `y` positive ganze Zahlen sind, dürfen den regulären Multiplikationsoperator `*` jedoch nicht verwenden.

```
def prod(x, y):
```

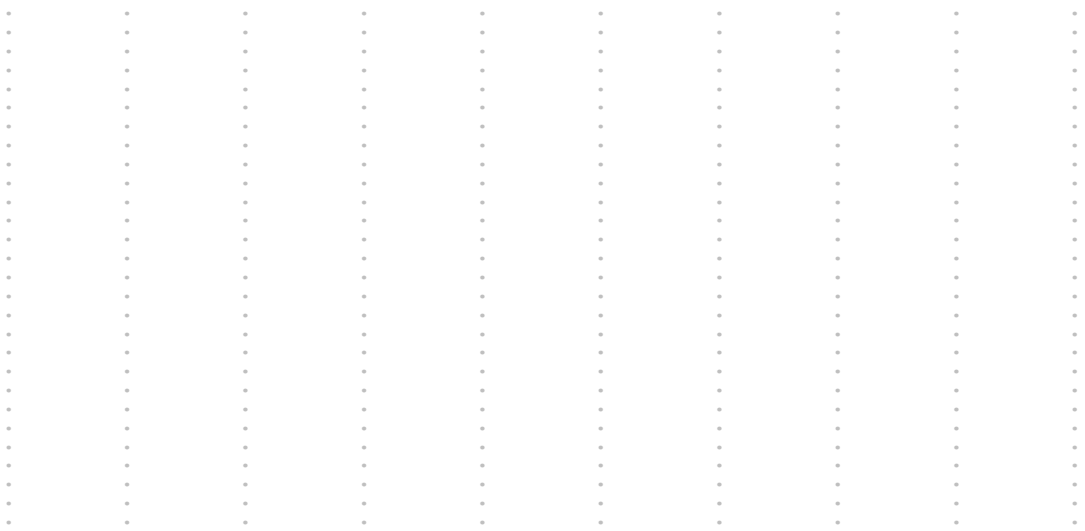


```
assert prod(2, 0) == 0
assert prod(5, 2) == 10
```

b) Invertieren einer Liste 8 Punkte

Implementieren Sie die Funktion `reverse`, welche *rekursiv* die Reihenfolge einer Liste umkehrt.

```
def reverse(l):
```



```
assert reverse([]) == []
assert reverse([2]) == [2]
assert reverse([2, 6, 5]) == [5, 6, 2]
```

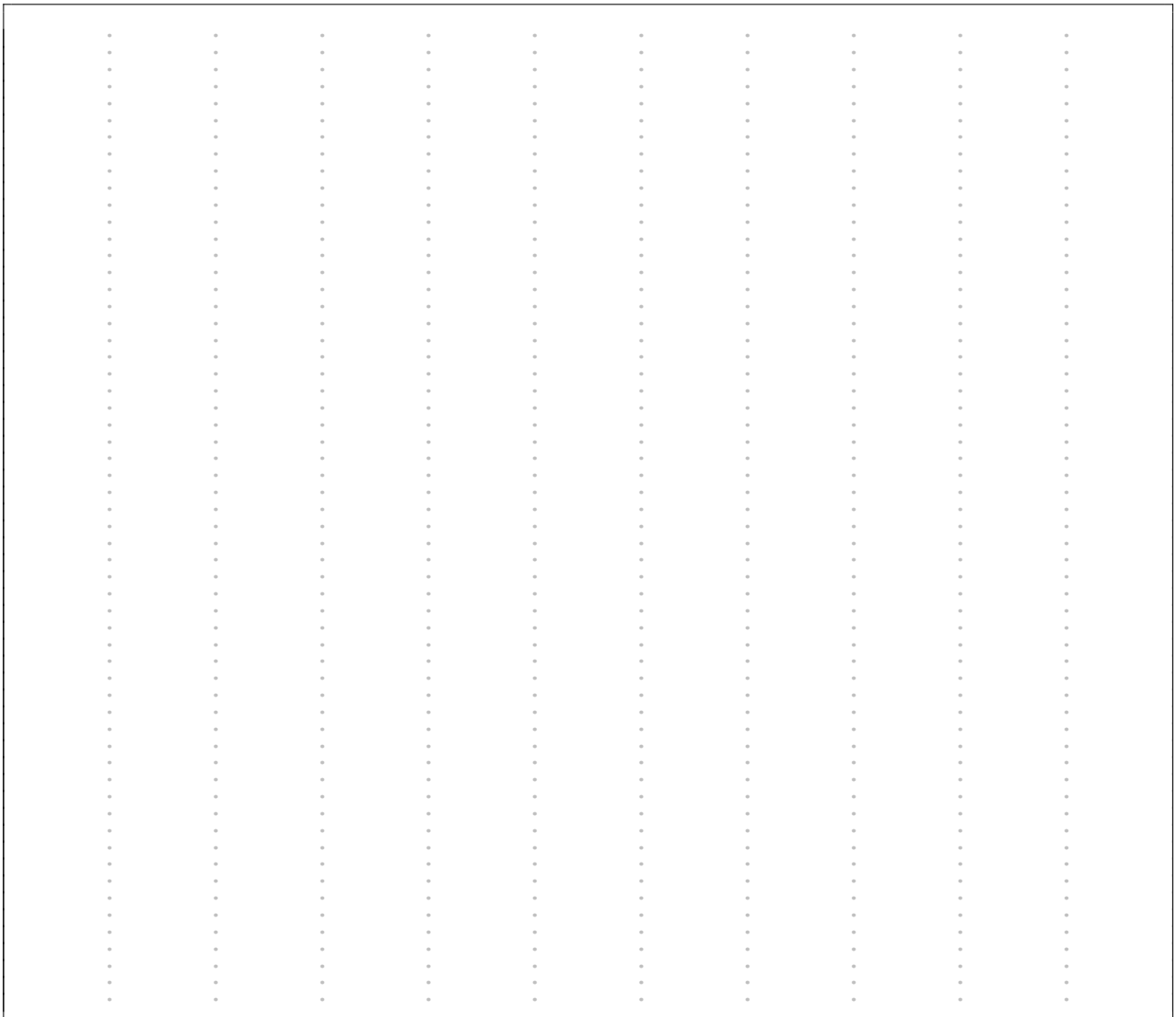
Aufgabe 4: Objekt-Orientierung & Tests 20 Punkte

In diesem Task werden Sie eine Klasse und die dazugehörigen Unittests implementieren. Benutzen Sie die Python Bibliothek `unittest` für Ihre Tests. Sie müssen keine `import` Statements angeben.

a) Accounting 8 Punkte

Definieren Sie eine Klasse `BankAccount`. Der Kontostand kann über `balance` abgefragt und mittels `deposit` und `withdraw` verändert werden. Neue Accounts haben eine `balance` von 0 allerdings wird im Konstruktor ein Kreditlimit übergeben. Ein Aufruf von `available` informiert über den maximalen Betrag, der abgehoben werden kann. Lösen Sie einen `AssertionError` aus (`raise`), wenn im Konstruktor ein negatives Kreditlimit angegeben wird oder wenn ein `withdraw` den erlaubten Kreditrahmen überschreitet.

Hinweis: `deposit` und `withdraw` haben keinen Rückgabewert und keine Ausgabe (`print`).



```
# example usage
acc = BankAccount(100)
print(acc.balance()) # prints '0'
print(acc.available()) # prints '100'
acc.deposit(30) # balance: 30, available: 130 (illustration, no "print")
acc.withdraw(40) # balance: -10, available: 90 (illustration, no "print")
acc.withdraw(91) # AssertionError
```

b) Black-Box Unittests 12 Punkte

Eine andere Person wird eine weitere `BankAccount` Implementierung erstellen, die derselben Spezifikation folgt. Erweitern Sie `TestCase` und erstellen Sie eine Testsuite, die die Korrektheit dieser Implementierung überprüft, ohne deren interne Umsetzung zu kennen. Sie müssen nicht alles erschöpfend testen, stellen Sie nur einen Test für den Konstruktor sowie Tests für jede der `BankAccount` Methoden bereit und testen Sie normale Benutzung. Überprüfen Sie zusätzlich in einem weiteren Test, dass übergrosse Abbuchungen (`withdraw`) einen `AssertionError` verursachen.

Hinweis: Benutzen Sie nicht das eingebaute Python Statement `assert`. Verwenden Sie stattdessen die asserts der `TestCase` Basisklasse.

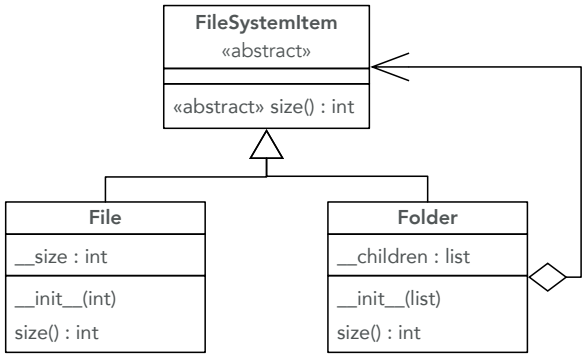
Hinweis: Überprüfen (`assertX`) Sie nicht mehr als eine Eigenschaft in einem Testfall.

Hinweis: Betrachten Sie diese Aufgabe als eigenständigen Teil, unabhängig von der vorigen Aufgabe. Beide werden unabhängig voneinander bewertet und sind jeweils ohne den anderen lösbar.

Aufgabe 5: Vererbung & Komposition 14 Punkte

Sie entscheiden sich dazu eine Abstraktion für Dateisysteminhalte zu erzeugen, um die Berechnung des benötigten Speicherplatzes auf einer Festplatte zu vereinfachen. Im folgenden ist die Abstraktion des Dateisystems als UML Diagram abgebildet, desweiteren finden Sie ein Beispiel, das die Verwendung der zu erstellenden Klassen spezifiziert. Die Abstraktion enthält das generische `FileSystemItem` und die beiden Spezialisierungen `File` und `Folder`.

Jedes `FileSystemItem` kann nach seiner Grösse gefragt werden. Die Grösse eines `Files` ist statisch und wird direkt bei der Instanzierung angegeben. Die Grösse eines `Folders` berechnet sich durch die Summe aller enthaltener Inhalte. Diese Inhalte werden in einer Liste an den Konstruktor übergeben, die sowohl `Files`, als auch verschachtelte `Folders` enthält.




```
assert File(1).size() == 1
assert Folder([]).size() == 0
assert Folder([File(2), \
  File(3)]).size() == 5
assert Folder([File(4), \
  Folder([File(5)])]).size() == 9
```

Im folgenden werden Sie die drei Klassen `FileSystemItem`, `File` und `Folder` erstellen. Sie müssen in Ihren Lösungen kein `import` angeben und brauchen nicht auf invalide Parameter überprüfen.

a) Implementiere die abstrakte Basisklasse `FileSystemItem` 4 Punkte

Erstelle die Klasse `FileSystemItem` als *abstrakte Basisklasse*. Erweitere dafür `ABC` und annotiere die Methode `size` mit `abstractmethod`, um eine Instanzierung der Klasse zu verhindern.

b) Implementiere `File` entsprechend der Spezifikation. 4 Punkte

c) Implementiere `Folder` entsprechend der Spezifikation. 6 Punkte

Aufgabe 6: Verwendung von Modulen — 10 Punkte

Moderne Computer enthalten eine Vielzahl von Sensoren, die den Systemzustand überwachen. Nehmen Sie an, Sie haben die folgende Datei `stats.py` gefunden, die eine Hilfsfunktion für den Zugriff auf verschiedene Metriken bereitstellt. Sie möchten eine bestimmte Funktion daraus in Ihrem Programm verwenden, im Header steht:

```
# content of file "stats.py"
def get_system_stats():
   '''Provides access to useful system stats (key -> description):
   - cpu_temp -> The temperature of the CPU sensor in "Kelvin" (float)
   - fan_speed -> The speed of the CPU fan in "rotations per minute" (int)
   - ... (several other stats, cut for brevity)
   Returns: A dictionary that contains all stats by key.'''
```

Definieren Sie die Klasse `TempReader`, die den Zugriff auf die CPU Temperatur vereinfacht. Die Klasse soll die beiden Methoden `celsius` und `fahrenheit` definieren, die einen formatierten String der aktuellen Temperatur in der entsprechenden Einheit zurückgibt.

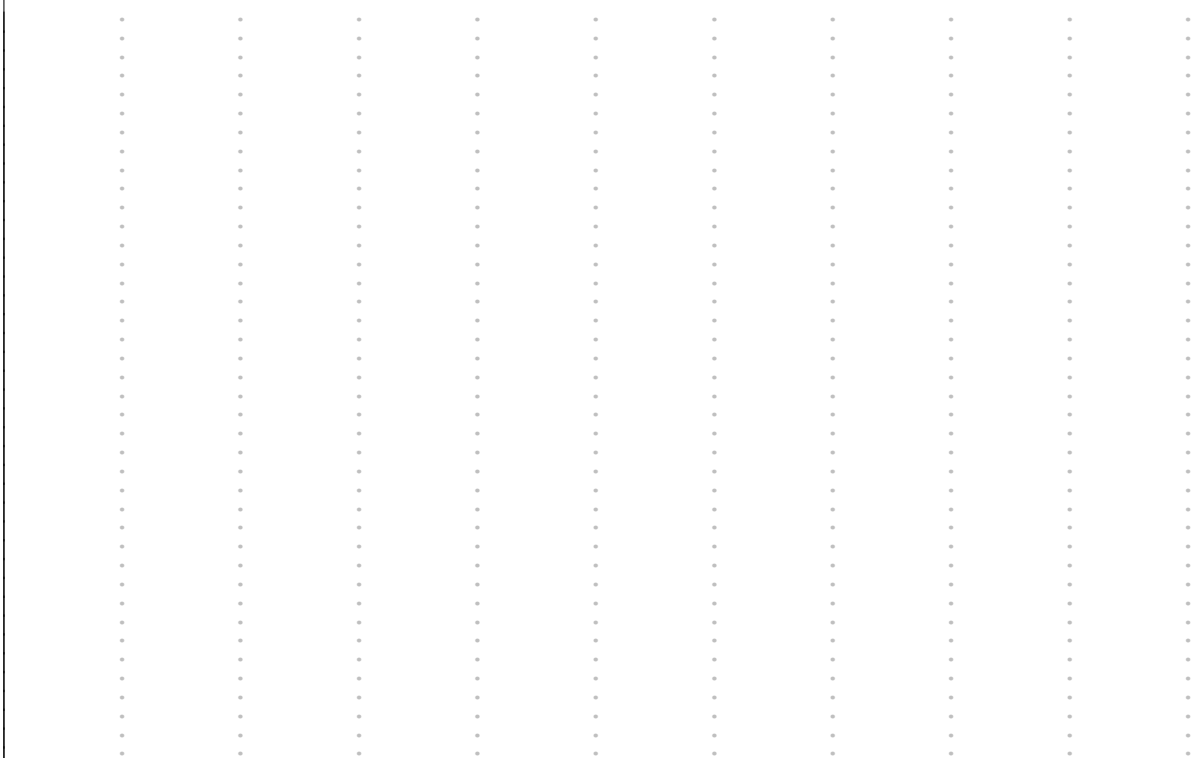
Leider gibt die Hilfsfunktion Temperaturen in Kelvin zurück (T_K), so dass der Wert erst in Celsius ($T_C = T_K - 273.15$) oder in Fahrenheit ($T_F = 1.8 \cdot T_K - 459.67$) konvertiert werden muss. Die ausgegebene Zahl soll auf eine Stelle nach dem Komma gerundet werden.

Hinweis: Wiederholte Aufrufe der `TempReader` Methoden sollen immer aktuelle Werte ausgeben!

Hinweis: Vergessen Sie nicht Ihrer Lösung `import` Statements hinzuzufügen.

Hinweis: Beide Dateien dieser Aufgabe liegen auf unterster Ebene des Modulsuchpfades.

```
# content of file "TempReader.py"
```



```
# expected behavior with example output
tr = TempReader()
print(tr.celsius()) # "56.2C"
print(tr.fahrenheit()) # "133.2F"
```

Nützliche Pythonfunktionen

Strings

str.upper() / str.lower() Gibt einen neuen String zurück, in dem alle Buchstaben zu *Gross-/Kleinbuchstaben* geändert wurden.

str.isupper() / str.islower() Gibt `True` zurück, falls alle Zeichen des nicht leeren Strings `str` Grossbuchstaben/Kleinbuchstaben sind, andernfalls `False`.

str.split(sep) Bricht einen Strings `str` bei jedem Vorkommen von `sep` in einzelne Wörter. `sep` ist optional, standardmässig werden die Wörter durch Whitespacezeichen getrennt (space, tab, newline, return, formfeed).

str.join(words) Erstellt einen String aus den Wörtern in `words` durch Aneinanderreihung. Die Wörter werden mit dem Wert von `str` verbunden.

str.isalpha() / str.isdigit() Ist `True`, wenn alle Zeichen eines nicht leeren Strings Buchstaben/Zahlen sind, sonst `False`.

str.startswith(prefix) Ist `True`, wenn der String `str` mit `prefix` beginnt, sonst `False`.

str.endswith(suffix) Ist `True`, wenn der String `str` mit `suffix` endet, sonst `False`.

string.find(x) Ermittelt den Startindex von `x`, wenn es im String vorkommt, sonst `-1`.

Lists

list.append(x) Hängt ein Element `x` an das Ende der Liste `a` an; äquivalent zu `a[len(a):] = [x]`.

list.remove(x) Entfernt das erste Element der Liste, dessen Wert `x` ist. Wirft einen Fehler, falls kein solches Element vorhanden ist.

list.index(x) Gibt den Index des ersten Elements zurück, dessen Wert `x` ist. Wirft einen Fehler, falls kein solches Element vorhanden ist.

list.count(x) Zählt wie häufig `x` in einer Liste vorkommt.

Dictionaries

key in dict `True`, wenn `key` im Dictionary vorhanden ist, sonst `False`.

dict.keys() Gibt eine Liste aller Keys zurück, die im Dictionary `dict` definiert sind.

dict.items() Gibt eine Liste aller `(Key, Value)` Tuples des Dictionary zurück.

dict.values() Gibt eine Liste aller Dictionary Values zurück.

dict.get(key, default=None) Gibt es den Wert zurück, der mit `key` assoziiert ist oder `default`, wenn der Key nicht existiert.

dict.pop(key) Entfernt `key` aus dem Dictionary und gibt dessen vorherigen Wert zurück.

Files

open(filename, 'r') Öffnet die Datei `filename` zum Lesen und gibt ein Dateiobjekt zurück.

open(filename, 'w') Öffnet die Datei `filename` zum Schreiben und gibt ein Dateiobjekt zurück.

f.close() Schliesst das Dateiobjekt `f`.

f.readline() Gibt die nächste Zeile des Dateiobjekts `f` zurück.

f.readlines() Gibt alle Zeilen des Dateiobjekts `f` zurück.

os.path.isfile(file) Ist `True`, wenn `file` existiert und eine reguläre Datei ist.

Other

isinstance(obj, type) Ist `True`, wenn der Typ von `obj` kompatibel zu `type` ist, ansonsten `False`.

len(obj) Gibt die Länge eines Objekts zurück. `obj` kann eine Sequenz sein (z.B.: string, list, etc.) oder eine collection (z.B.: dictionary).

sorted(sequence) Erzeugt aus den Elementen der Sequenz eine neue sortierte Liste.

TestCase

assertEqual(a, b) Testet, dass `a` und `b` gleich sind. Ist dies nicht der Fall, schlägt der Test fehl.

assertTrue(a) / assertFalse(a) Testet, dass `a` den Wert `True` / `False` hat.

assertRaise(Type) Kann in einem `with` Statement verwendet werden, um zu testen, dass der umschlossene Quelltext den angegebenen Fehlertypen `raised`. Falls nicht, schlägt der Test fehl.