Python – Übung 7

1 Vererbung

Die Klasse `Pet`, mit welcher man ein Haustier abbilden kann, ist in der Datei `pet.py` implementiert. Bei der Instanziierung muss der Name und das Geburtsdatum des Haustiers angegeben werden, z.B.:

```
haustier = Pet(name="Bobby", date_of_birth="1.1.2019")
```

Mit dem Property `age` kann das Alter (in Jahren) des Haustiers ermittelt werden, z.B.:

```
>>> print(f"{haustier.name} ist {haustier.age:.1f} Jahre alt.")
Bobby ist 1.2 Jahre alt.
```

☞ Implementieren Sie die beiden Klassen `Dog` und `Hamster` als Subklassen der Klasse `Pet`. Überschreiben Sie jeweils das Property `age`, so dass bei diesen Tieren das Alter in Menschenjahre umgerechnet wird, indem das tatsächliche Alter mit einem tierabhängigen Faktor multipliziert wird. Die Faktoren sind: 5.5 für den Hund und 39 für den Hamster. Implementieren Sie zudem für den Hund die Methode `woof()`, welche beim Aufruf den String `"bow-wow!"` auf der Konsole ausgibt.

```
>>> hund = Dog("Lassie", "1.1.2019")
>>> print(f"{hund.name} ist {hund.age:.1f} Jahre alt.")
Lassie ist 6.8 Jahre alt.
```

```
>>> hamster = Hamster("Daisy", "1.1.2019")
>>> print(f"{hamster.name} ist {hamster.age:.1f} Jahre alt.")
Daisy ist 48.4 Jahre alt.
```

```
>>> hund.woof()
bow-wow!
```

2 Wanduhr

Die mitgelieferte Python-Datei `time_and_date.py` enthält zwei Klassendefinitionen:

Clock-Klasse Sie besitzt drei *protected* Instanzvariablen: `_seconds`, `_minutes` und `_hours`, dessen Werte bei der Instanziierung gesetzt werden. Die *public* Methode `time_increment()` inkrementiert die Uhrzeit um eine Sekunde. Mit der magischen Methode `__str__()` wird die Uhrzeit als String im folgenden Format zurückgegeben: `HH:MM:SS`, z.B.:

```
>>> c = Clock(12, 30, 59) # oder Clock(hours=12, minutes=30, seconds=59)
>>> print(c)
12:30:59
```

Calendar-Klasse Sie besitzt drei *protected* Instanzvariablen: `_day`, `_month` und `_year`, deren Werte bei der Instanziierung gesetzt werden. Die *public* Methode `date_increment()` inkrementiert das Datum um einen Tag. Mit der magischen Methode `__str__()` wird das Datum als String im folgenden Format zurückgegeben: `dd.mm.yyyy`, z.B.:

```
>>> d = Calendar(1, 4, 2020) # oder Calendar(day=1, month=4, year=2020)
>>> print(d)
01.04.2020
```

☞ Implementieren Sie die Subklasse `CalendarWallClock`, die von den beiden anderen Klassen `Calendar` und `Clock` erbt und deren Funktionalität, wie unten beschrieben, kombiniert.

CalendarWallClock-Klasse Sie erbt von den beiden anderen Klassen `Clock` und `Calendar`. Mit der *public* Methode `time_increment()`, welche die gleichnamige Methode der `Clock`-Klasse überschreibt, soll die Uhrzeit um eine Sekunde inkrementiert werden. Beim Wechsel von 23:59:59 auf 00:00:00 soll automatisch das Datum um einen Tag inkrementiert werden. Mit der eigenen magischen Methode `__str__()` soll das Datum und die Uhrzeit als String im folgenden Format zurückgegeben werden: `dd.mm.yyyy - HH:MM:SS`.

Hinweis: Nehmen Sie die bestehenden Methoden der beiden Superklassen zur Hilfe.

```
>>> wallclock = CalendarWallClock(28, 2, 2020, 23, 59, 59)
>>> wallclock.time_increment()
>>> print(wallclock)
29.02.2020 - 00:00:00
```

```
>>> wallclock = CalendarWallClock(day=28, month=2, year=2019, hours=23,
   minutes=59, seconds=59)
>>> wallclock.time_increment()
>>> print(wallclock)
01.03.2019 - 00:00:00
```

3 Aufgaben aus dem Buch

☞ Lösen Sie folgende Aufgaben aus dem Buch.

Kapitel	Seiten	Aufgaben
29 – Aufgaben	Buch: 302, PDF: 311	4

4 Sortierte Liste

☞ Implementieren Sie die Klasse `SortedList`, welche eine Subklasse der eingebauten Klasse `list` sein soll und deren Elemente nach jeder Operation (z.B. Hinzufügen von Elementen) automatisch wieder in aufsteigender Reihenfolge sortiert.

```
>>> s = SortedList([3, 1, 2])
>>> print(s)
[1, 2, 3]
```

```
>>> s = SortedList()
>>> s.append(6)
>>> s.extend([5, 3, 4])
>>> print(s)
[3, 4, 5, 6]
```

Hinweis: Überschreiben Sie jede Methode, die ein oder mehrere Elemente in die Liste einfügt oder modifiziert, so dass nach der jeweiligen Operation die Liste gleich sortiert wird. Die betroffenen Methoden sind in Tab. 1 aufgelistet. Neutralisieren Sie zudem die Wirkung der Methoden `reverse()` und `sort()`.

Tabelle 1: `list`-Methoden, die ein oder mehrere Elemente einfügen oder modifizieren.

Methode	Beispiel-Code	Beschreibung
`__setitem__(i, x)`	`a[i] = x`	Das `i`-te Element mit `x` überschreiben.
`__iadd__(b)`	`a += b`	Liste `a` mit der Liste `b` erweitern.
`__imul__(n)`	`a *= n`	Liste `a` n-mal vervielfachen.
`append(x)`	`a.append(x)`	Element `x` an Liste `a` anhängen.
`extend(b)`	`a.extend(b)`	Liste `a` mit der Liste `b` erweitern.
`insert(i, x)`	`a.insert(i, x)`	Element `x` an der Position `i` einfügen.
`reverse()`	`a.reverse()`	Reihenfolge umkehren.
`sort([key, reverse])`	`a.sort()`	Elemente sortieren, opt.: `key`, `reverse`.

5 Telefonbuch

☞ Implementieren Sie die Klasse `PhoneBook`, welche die Funktionalität eines Telefonbuches bieten soll. Die Klasse soll Personennamen und deren Telefonnummer speichern und verwalten, wobei beide Angaben als Strings angegeben werden. Die Klasse soll wie folgt benutzt werden können:

PhoneBook(data) instantiiert ein neues Objekt und initialisiert die internen Daten mit dem angegebenen Dictionary, falls vorhanden, z.B.:

```
>>> d = {"Alice": "004102", "Carol": "004101", "Ted": "004103"}
>>> pb = PhoneBook(d)
```

Das Dictionary soll kopiert und nicht einfach referenziert werden, denn es könnte sonst ausserhalb des Klassenobjektes darauf zugegriffen und modifiziert werden. Benutzen Sie ein *protected* Attribut für die internen Daten.

PhoneBook.add(self, name, number): speichert den angegebenen Namen mit der Telefonnummer in den internen Daten ab. Falls der Eintrag schon existiert, wird es überschrieben.

```
>>> pb.add("Boss", "004101")
```

PhoneBook.remove(self, name): entfernt den angegebenen Namen aus den internen Daten:

```
>>> pb.remove("Ted")
```

PhoneBook.__str__(self): implementiert die magische Methode, um das Objekt in einen String umzuwandeln, z.B. wenn das Objekt in der `print()`-Funktion angegeben wird. Die Methode soll ein String mit den Einträgen in alphabetischer Reihenfolge zurückgeben, z.B.:

```
>>> print(pb)
Alice: 004102
Boss: 004101
Carol: 004101
```

PhoneBook.__contains__(self, term): implementiert die magische Methode, welche automatisch vom `in`-Operator aufgerufen wird. Die Methode soll `True` zurückgeben, falls der angegebene Begriff mit einem Namen oder einer Telefonnummer übereinstimmt, sonst soll sie `False` zurückgeben, z.B.:

```
>>> "Alice" in pb
True
```

PhoneBook.__getitem__(self, term): implementiert die magische Methode, welche automatisch aufgerufen wird, wenn ein Element indexiert wird. Die Methode liefert ein Dictionary mit allen Treffern zurück, wo entweder dessen Name oder Nummer mit dem Begriff übereinstimmt. Bei keiner Übereinstimmung wird ein leeres Dictionary zurückgegeben.

```
>>> pb["004101"]
{"Carol": "004101", "Boss": "004101"}
>>> pb["Alice"]
{"Alice": "004102"}
```