University of Zurich[UZH]

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

Informatik I
Einführung in die Programmierung

Assessment-Prüfung HS19

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 12 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.
- Benutzen Sie **Python 3.7** und die entsprechenden Funktionen. 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: | | | | | | | | | | | | | | | | |

Matrikelnr.: | | - | | | | - | | |

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

Aufgabe 1: Allgemeine Fragen — 20 Punkte

Diese Aufgabe enthält einige kurze Pythonsnippets, die mit einem Ausdruck enden. Schreiben Sie den *Typ* und den *Wert* des Ausdrucks in die vorgesehenen Felder. Denken Sie daran, dass Ausdrücke ohne *expliziten* Wert auch einen *impliziten* Typ und Wert haben. Sollte die Ausführung mit Fehler abbrechen, dann wählen Sie *Exception* als Typ und schreiben *error* als Wert. Wenn das Snippet eine Endlosschleife ausführt, wählen Sie *NoneType* als Type und *endless loop* als Wert.

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

Hinweis: Die Snippets sind getrennt zu betrachten und haben keine Seiteneffekte aufeinander.

a) 2 Punkte

```
not ()
```

Typ: Wert:

b) 2 Punkte

```
print("Hello World!")
```

Typ: Wert:

c) 2 Punkte

```
input("How tall are you?") # nehmen Sie an, die Eingabe wäre 1.83
```

Typ: Wert:

d) 2 Punkte

```
l = [1, 2, 3, 4]
l[1:3] = []
l
```

Typ: Wert:

e) 2 Punkte

```
def fun(l):
    if len(l) > 0:
        return fun(l[0])
    else:
        return 42
l = []
l.append(l)
fun(l)
```

Typ: Wert:

f) 2 Punkte

```
class Person:
    def get_name(self):
        return self.name
p1 = Person("Adam")
p2 = Person("Bran")
p1.get_name()
```

Typ: Wert:

g) 2 Punkte

```
class X: pass
class Y(X): pass
1 if isinstance(Y(), object) else 2.3
```

Typ: Wert:

h) 2 Punkte

```
a = 2
b = 3.0
assert a < b
a*b
```

Typ: Wert:

i) 2 Punkte

```
try:
   x = None
   raise IndexError()
   x = 1
except IndexError:
   x = 2.0
except:
   x = True
else:
   x = -1+0j
finally:
   x = "fin"
x
```

Typ: Wert:

j) 2 Punkte

```
try:
   x = 42 / 0
finally:
   x = 1
x
```

Typ: Wert:

Aufgabe 2: Hailstone Sequenz — 10 Punkte

Schreiben Sie eine Funktion, die startend von einem beliebigen *positiven* Integer `n` eine Integer-Liste generiert. Die Liste soll mit `n` starten und mit zwei Regeln weitergeführt werden:

- ist das aktuelle Element gerade, teile durch 2, um das nächste Element zu generieren
- ist das aktuelle Element ungerade, multipliziere mit 3 und addiere 1 für das nächste Element

Die Sequenz endet, sobald `1` erreicht wurde, um eine endlose Weiterführung zu verhindern (1, 4, 2, 1, ...). Die resultierende Sequenz wird auch als *Hailstone Sequenz* bezeichnet.

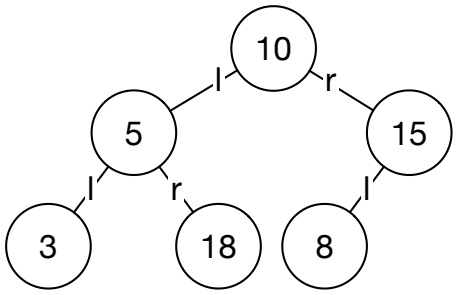
```
def hailstone(n):
```

```
assert hailstone(1) == [1]
assert hailstone(3) == [3, 10, 5, 16, 8, 4, 2, 1]
assert hailstone(7) == [7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, ...]
```

Aufgabe 3: Rekursion — 10 Punkte

Ein Binärbaum ist eine Datenstruktur, in der jede `Node` einen Wert und zwei (optionale) Kinder besitzt, welche als linkes und rechtes Kind bezeichnet werden. `root` wird neben dem Code visualisiert.

```
class Node:
   def __init__(self, v, l=None, r=None):
      self.v = v
      self.l = l
      self.r = r
root = Node(10, \
  Node(5, Node(3), Node(18)), \
  Node(15, Node(8)))
```



Implementieren Sie die Funktion `range_sum`, die für einen Binärbaum und die zwei Grenzen `lower` und `upper` die Summe aller im Baum enthaltener Werte `v` berechnet, für die `lower <= v < upper`.

```
def range_sum(node, lower, upper):
```

```
assert range_sum(Node(7), 1, 100) == 7
assert range_sum(Node(2, Node(3, Node(4))), 2, 4) == 5
assert range_sum(root, 4, 10) == 13 # see example above
```

Aufgabe 4: Objekt-Orientierung & Tests 20 Punkte

Implementieren Sie die beiden Klassen `Backpack` und `Item`, die bei der nächsten Campingplanung helfen. Ein `Backpack` hat ein maximales `volume` (in Litern), ein `Item` einen `name` und ein `volume` (in Litern). Diese Parameter werden im Konstruktor gesetzt. Bitte beachten Sie das folgende Beispiel:

```
bp = Backpack(4.0)
bp.pack(Item("water bottle", 0.75))
bp.pack(Item("lighter", 0.05))
bp.current_volume() # 0.755
bp.unpack() # returns Item("lighter", 0.05)
bp.pack(Item("camping tent", 20.0)) # AssertionError!
```

Implementieren Sie die drei `Backpack`-Methoden des Beispiels: 1) Ein Benutzer kann ein `Item` ein`pack`en, werfen Sie einen `AssertionError` wenn der Platz nicht ausreicht. 2) `current_volume` berechnet das Gesamtvolumen der gepackten `Item`s. 3) `unpack` entfernt das zuletzt hinzugefügte `Item` und gibt es zurück oder `None`, wenn der Rucksack leer ist.

Hinweis: Die Implementierung muss nicht auf inkorrekte Typen oder invalide Werte prüfen.

Hinweis: Imports für `Backpack`, `Item`, und `TestCase` sind nicht erforderlich.

a) Implementierung von `Backpack` 10 Punkte

b) Implementierung von `Item` 2 Punkte

c) Unit Testing 8 Punkte

Schreiben Sie eine Testsuite für `Backpack`, die eine beliebige Implementierung auf die Spezifikation überprüft. Nutzen Sie Python's `unittest` Modul und erweitern Sie `TestCase`. Die Testsuite muss nicht umfassend sein, aber stellen Sie einen Test für den Konstruktor bereit, jeweils einen für die drei Methoden, und einen, der verifiziert, dass zu grosse Items einen `AssertionError` verursachen.

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

Aufgabe 5: Vererbung & Komposition 20 Punkte

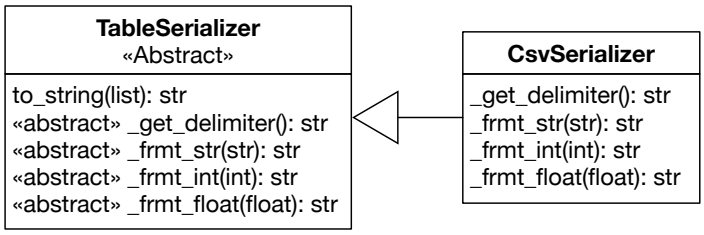
Tabellarische Daten sind Einträge (Zeilen) mit Attributen (Spalten) die in Standardformaten gespeichert werden. Ein solches Format ist .csv, in dem jeder Eintrag als Zeile gespeichert wird, in der die Attribute mit Komma getrennt sind. Im Folgenden sehen Sie ein Beispiel. Die Tabelle ist links, die .csv Darstellung in der Mitte und die programmatische Darstellung (Liste von Tuplen) rechts.

Name	Age	Size
Hans	53	1.78
Frieda	27	1.63

```
"Name","Age","Size"
"Hans",53,1.78
"Frieda",27,1.63
```

```
[("Name","Age","Size"),
 ("Hans",53,1.78),
 ("Frieda",27,1.63)]
```

Es gibt viele Variationen bei der Formatierung: Zum Beispiel der Einsatz eines Tabs ("\t") als Trenner, die Vermeidung von Hochkommas bei Strings oder die Benutzung spezifischer Zahlenformate. In diesem Task implementieren Sie die Hierarchie der abstrakten Basisklasse `TableSerializer`, die erweiterbar ist, um verschiedene Formate wie das Beispiel .csv zu unterstützen.



Der abstrakte `TableSerializer` implementiert `to_string`, das über die Tabelle iteriert, um die entsprechende Stringdarstellung zu erzeugen. Erweiterbarkeit wird durch die Delegation der Formatierung an die abstrakten Methoden `get_delimiter` und `frmt_...` erreicht. Subklassen müssen diese implementieren und können dann, zum Beispiel, in der Implementierung von `frmt_float` entscheiden, ob 1.234 als `"1.234"` oder `1.234` oder `1.2` (gerundet) or ganz anders dargestellt wird.

Hinweis: Sie können für diesen Task annehmen, dass Attribute stets `str`, `int` oder `float` sind.

Hinweis: `TableSerializer` ist eine *abstrakte Basisklasse*. Erweitern Sie `ABC` und annotieren Sie abstrakte Methoden mit `abstractmethod`, um die Instanzierung der Klasse zu verhindern. Sie können die notwendigen `import`s weglassen.

a) Implementieren Sie `TableSerializer` entsprechend der UML Spezifikation 10 Punkte

b) Bereitstellung einer konkreten Subklasse 8 Punkte

Implementieren Sie eine Subklasse, `CsvSerializer`, die dem .csv Format aus dem Beispiel folgt.

c) Benutzung der Hierarchie 2 Punkte

Instanzieren Sie `CsvSerializer`, formatieren Sie `table` und `print`en Sie es auf der Konsole.

```
table = (("Name","Age","Size"), ("Hans",53,1.78), ("Frieda",27,1.63))
```

Aufgabe 6: Verwendung von Modulen — 10 Punkte

Implementieren Sie eine neue geo-location-basierte App, die den nächsten Bahnhof finden kann. Nehmen Sie an, die Bibliothek `navigation.py` stellt dafür folgende Funktionen bereit.

```
# content of file "navigation.py"
def get_current_position():
   '''Returns the current GPS coordinates (latitude, longitude) in a string like
   "43.63871944444445,-116.2413513485235".'''
def find_train_stations(position):
   '''Given the current position in a tuple of two floats (latitude, longitude),
   returns all stations in a 5km radius. The stations will be returned in a list
   of tuples with the format (str, (float, float)). The first element is the
   station name, the second is the exact position tuple. The values are ordered
   by distance (closest first). The list is empty if no stations are nearby.'''
```

Implementieren Sie die Funktion `find_next_station`, die die Bibliothek nutzt, um den nächsten Bahnhof zu finden. Die Funktion soll den Bahnhofnamen als String zurückgeben oder `None`, wenn keiner in der Nähe ist. Wiederholte Aufrufe sollen die aktuelle Nutzerposition berücksichtigen.

Hinweis: Denken Sie an die `import`s. Alle Dateien liegen auf unterster Ebene des Modulsuchpfades.

```
def find_next_station():

print(find_next_station()) # for example, "Bahnhof Oerlikon"
```

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`.

string.replace(old, new) for Seb: write down here in german

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.