Aufgabe zur Objektorientierung in Python

Aufgabe 1

Erstelle eine Klasse „Bankkonten“. Überlege dir sinnvolle Attribute und Methoden, die man für ein Bankkonto braucht. Das dauert. Wirklich. Man muss sich in den Schmonz ziemlich reindenken und viel probieren!

Das Ziel für heute: Du kannst Geldbeträge zwischen unterschiedlichen Konten verschieben (dort abbuchen, da aufbuchen). Das lässt sich mit dem Wissen, was du hast, elegant und weniger elegant lösen.

Die harte Variante: Du kannst einer Methode eines Objekts auch ein anderes Objekt übergeben (das ist aber schon die ganz harte Variante für den Anfang, an dem du stehst).

Klicke hier für einen Lösungsansatz

account.py

```
# Thx an Aaron für große Teile des Codes!

class konto:
    # Konstruktor, der uns das neue Konto baut
    def __init__(self, nummer, kontostand):
        self.nummer = nummer
        self.kontostand = kontostand

    # Methode zum Überweisen
    # Erwartet: value (Summe, die übertragen werden soll)
    # Erwartet: target (Zielkonto)
    # Auf den eigenen Kontostand können wir mit self.kontostand
zugreifen
    # Auf den Zielkontostand können wir mit target.kontostand zugreifen
    # target haben wir im Hauptprogramm übergeben

    def ueberweisen(self, value, target):
      self.kontostand = self.kontostand - value
      target.kontostand = target.kontostand + value

konto1 = konto(1, 1000)
konto2 = konto(2, 2000)

print("Alter Kontostand:", konto1.kontostand)
print("Alter Kontostand:", konto2.kontostand)

# Wir greifen auf die Methode "überweisen" innerhalb der Klasse konto
zu.
# Vor dem Punkt steht das Objekt, das wir verändern wollen
```

```
konto1.ueberweisen(100, konto2)

print("Neuer Kontostand:", konto1.kontostand)
print("Neuer Kontostand:", konto2.kontostand)
```

Aufgabe 2

Bankkonten sind eine komplexe Sache. Erwachsene dürfen z.B. im Rahmen eines Dispositionskredits ihr Konto um einen bestimmten Betrag „überziehen“. Spezielle Jugendkonten bieten den kompletten Funktionsumfang, verbieten aber jedwede Überziehung.

Überlege dir, wie du Dispositionskredite und Jugendkonten in deiner Kontoklasse realisieren kannst. Die Dispositionskredite sollen sich durch eine Methode frei wählen lassen, jedoch nie über 50% des aktuellen Kontostandes, d.h. das Konto muss beim Einrichten im Plus sein.

Wenn ein Jugendlicher jedoch versucht, sich einen solchen Kredit einzuräumen, soll diese Aktion scheitern. Jugendliche sollen ihr Konto zudem nicht überziehen können.

Klicke hier für einen Lösungsansatz

jugend.py

```
class konto:
    # Konstruktor, der uns das neue Konto baut
    def __init__(self, nummer, kontostand, isAdult, dispo):
        self.nummer = nummer
        self.kontostand = kontostand
        self.isAdult = isAdult
        self.dispo = dispo

    def ueberweisen(self, value, target):
        self.kontostand = self.kontostand - value
        target.kontostand = target.konto

    def setDispo(self,value):
        if self.isAdult == 0:
            print("Du bist zu jung!")
            return
        if value > self.kontostand/2:
            print("Zu wenig Guthaben!")
            return
        else:
            self.dispo = value
            print("Dispo eingerichtet!")

# kundenkonto[0] hat die Nummer 1 und 1000 Euro Guthaben
```

```
# kundenkonto[1] hat die Nummer 2 und 2000 Euro Guthaben

kundenkonten = [konto(1,1000,0,0), konto(2,2000,1,0)]

# Geht schief, weil Jugendkonto
kundenkonten[0].setDispo(500)

# Geht schief, weil Deckung nicht ausreicht
kundenkonten[1].setDispo(2000)

# klappt
kundenkonten[1].setDispo(500)
```

Aufgabe 3

Auch die Bank braucht hin und wieder Informationen über ihre Kunden. So sollen Mitarbeitende z.B. einfach herausfinden können, wie viele Jugendkonten und Erwachsenenkonten es gerade gibt.

Das kannst du z.B. alles über eine Methode im Hauptprogramm realisieren.

Wenn du das elegant mit einer Schleife machen möchtest, musst du ja durch alle Konten laufen. Um die Konten in einer Schleife durchzählen zu können, wäre es ja hübsch, sie über einen Index ansprechen zu können, z.B. **kundenkonto[0].getDispo()** oder **kundenkonto[0].isAdult**.

Du kannst dir dazu ein Array von Objekten anlegen. Die Elemente sind durch ein Komma getrennt. In runden Klammern stehen hinter jedem Element die Basiswerte. Das wirst du ggf. an deine Klasse anpassen müssen. Die Umsetzung ist recht einfach:

array_of_objects.py

```
class konto:
    # Konstruktor, der uns das neue Konto baut
    def __init__(self, nummer, kontostand):
        self.nummer = nummer
        self.kontostand = kontostand

# kundenkonto[0] hat die Nummer 1 und 1000 Euro Guthaben
# kundenkonto[1] hat die Nummer 2 und 2000 Euro Guthaben

kundenkonten = [konto(1,1000), konto(2,2000)]
print(kundenkonto[1].kontostand)
```



Zur Erinnerung

len(array) gibt die die Anzahl der Elemente in einem Array zurück.

Klicke hier für einen Lösungsansatz

account_count.py

```
class konto:
    # Konstruktor, der uns das neue Konto baut
    def __init__(self, nummer, kontostand, isAdult, dispo):
        self.nummer = nummer
        self.kontostand = kontostand
        self.isAdult = isAdult
        self.dispo = dispo

    def ueberweisen(self, value, target):
        self.kontostand = self.kontostand - value
        target.kontostand = target.konto

    def setDispo(self,value):
        if self.isAdult == 0:
            print("Du bist zu jung!")
            return
        if value > self.kontostand/2:
            print("Zu wenig Guthaben!")
            return
        else:
            self.dispo = value
            print("Dispo eingerichtet!")

# kundenkonto[0] hat die Nummer 1 und 1000 Euro Guthaben
# kundenkonto[1] hat die Nummer 2 und 2000 Euro Guthaben

kundenkonten = [konto(1,1000,0,0), konto(2,2000,1,0)]

# Zählvariablen
youth_count = 0
adult_count = 0

for i in range(0,len(kundenkonten)):

    if (kundenkonten[i].isAdult == 1):
        adult_count += 1
    else:
        youth_count += 1

print(f'Es gibt {youth_count} Jugendkonten und {adult_count}
Erwachsenenkonten.')
```

Aufgabe 4

Der Bankvorstand hat sich gemeldet. Die Summe aller positiven Kontostände soll angezeigt werden können sowie die Höhe aller gerade laufenden Dispositionskredite. Außerdem soll berechnet werden können, ob die Bank selbst im Plus oder im Minus ist.

Klicke hier für einen Lösungsansatz

sum_accounts.py

```
class konto:
    # Konstruktor, der uns das neue Konto baut
    def __init__(self, nummer, kontostand, isAdult, dispo):
        self.nummer = nummer
        self.kontostand = kontostand
        self.isAdult = isAdult
        self.dispo = dispo

    def ueberweisen(self, value, target):
        self.kontostand = self.kontostand - value
        target.kontostand = target.konto

    def setDispo(self,value):
        if self.isAdult == 0:
            print("Du bist zu jung!")
            return
        if value > self.kontostand/2:
            print("Zu wenig Guthaben!")
            return
        else:
            self.dispo = value
            print("Dispo eingerichtet!")

# kundenkonto[0] hat die Nummer 1 und 1000 Euro Guthaben
# kundenkonto[1] hat die Nummer 2 und 2000 Euro Guthaben

kundenkonten = [konto(1,1000,0,0), konto(2,2000,1,1000)]

# Variablen zum Aufsummieren
sum_credit = 0
sum_dispo = 0

for i in range(0,len(kundenkonten)):
    sum_credit += kundenkonten[i].kontostand
    sum_dispo += kundenkonten[i].dispo

print(f'Die Guthabenhöhe aller Konten beträgt {sum_credit} Euro und die
Höhe der Dispokredite beträgt {sum_dispo} Euro. Der Guthabenstand der
Bank beträgt insgesamt {sum_credit} Euro.')
```

Aufgabe 5

Bei jeder Überweisung soll nun eine Gebühr von 1,5% des Überweisungsbetrages erhoben werden - außer bei Jugendkonten.

Klicke hier für einen Lösungsansatz

Bei Erwachsenenkonten ziehen wir bei jeder Überweisung 101,5% vom Konto ab, bei Jugendkonten nicht. Problem: Momentan verschwindet die Gebühr einfach im Nirgendwo. Hast du Ideen?

account_fee.py

```
def ueberweisen(self, value, target):
    if (self.isAdult == 1):
        self.kontostand = self.kontostand - value * 1.015
    else:
        self.kontostand = self.kontostand
    target.kontostand = target.konto
```

Aufgabe 6

Es gibt ein ernstes Problem mit der Methode zur Überweisung. Wenn der Kunde negative Überweisungsbeträge eingibt, bekommt er Geld **vom** anderen Konto. Das soll unterbunden werden.

Klicke hier für einen Lösungsansatz

handling_negative.py

```
def ueberweisen(self, value, target):
    if (value < 0):
        print("Nicht autorisierte Anforderung!")
        return
    if (self.isAdult == 1):
        self.kontostand = self.kontostand - value * 1.015
    else:
        self.kontostand = self.kontostand - value
    target.kontostand = target.konto + value
```

Aufgabe 7

Die Schulbücherei braucht deine Hilfe. Sie möchte weg von den ganzen Zettelkästen im Verleih. Die Schüler:innen sollen Bücher nach Kriterien suchen können. Sie dürfen zudem bis zu drei Bücher

ausleihen, wenn diese Bücher nicht schon verliehen sind. Überlege dir eine geeignete Klasse „Buch“ und eine geeignete Klasse „Schüler“. Welche Attribute und Methoden brauchen diese Klassen mindestens? Teste deine Klassen mit Beispielsuchen und Beispielausleihvorgängen.

- Du erledigst diese Aufgabe mit einem Partner / einer Partnerin.
- Die ausgeliehenen Bücher sollen innerhalb einer Person gespeichert werden. Wenn eine Person ausgewählt wird, sollen die von ihr entliehenen Bücher angezeigt werden.
- Verzettelt euch nicht! Realisiert nur die Attribute und Methoden, die ihr unbedingt braucht, um Basisfunktionen bereitzustellen
- Überlegt bitte einmal, wie ihr Aufgaben im Team verteilen könnt: Es müssen zwei Klassen geschrieben werden, es müssen Objekte zum Testen erzeugt werden, Methoden müssen getestet werden etc..

Hinweise

Da nur drei Bücher entliehen werden sollen, kann man es sich einfach machen und einfach drei Attribute für Bücher in der Klasse „Person“ hinterlegen. Jedes Atrribut zeigt dann auf ein Buchobjekt - oder eben nicht, wenn keins entliehen ist. Schicker ist es, die Buchobjekte in einem Array von Objekten zu speichern. Damit ist man später flexibler bei der Anzahl der entleihbaren Bücher und braucht auch im Prinzip weniger Speicherplatz.

Ein Element anhängen:

append.py

```
array = [0,1,2,3]
array.append(4)
for x in array:
   print (x)
```



Mehrere Elemente anhängen:

append_several.py

```
array = [0,1,2,3]
array.extend([4,5,6,7])
for x in array:
   print (x)
```

Ein Element löschen:

remove.py

```
array = [0,1,2,3]
array.remove(2)
for x in array:
```

```
    print (x)
```

Denke daran, dass du bei einem Array von Objekten nicht mit Werten herumschiebst, sondern mit Objekten, z.B. person[0], book[8].

remove.py



```
# Wir nehmen an, dass die Klasse für Personen "person" heißt und N Attribute hat
array = [person(Attribut1, Attribut2,...,AttributN),
person(Attribut1, Attribut2,...,AttributN),
person(Attribut1, Attribut2,...,AttributN), ... ]

# Entfernt person[0] aus dem Array
array.remove(person[0])
```

Klicke hier für einen Lösungsansatz

library.py

```
class book:
    def __init__(self, id, title, author, isAvailable):
        self.id = id
        self.title = title
        self.author = author
        self.isAvailable = isAvailable

    def rent():
        self.isAvailable = False

    def release():
        self.isAvailable = True

class user:
    def __init__(self, id, name, surename, hasBooks):
        self.id = id
        self.name = name
        self.surename = surename
        self.hasBooks = hasBooks

# Some Testing
book_test = book(1, "Harry Potter - Stein der Weisen", "Joanne K. Rowling", False)
print(book_test.title)
```

```
user_testHasBooks = []
user_test = user(1, "Maik", "Riecken", user_testHasBooks)
print(user_test.surename)
```

Aufgabe 8

Solarmodule sind in aller Munde. Diese bestehen aus einzelnen Solarzellen. Diese haben einen Spannung von ca. 0,6 Volt. Damit eine Solarzelle also 6V liefert, müssen 10 Solarzellen zu einem Modul zusammengeschaltet werden. Wir nehmen vereinfachend an, dass eine einzelne Solarzelle immer 1A an Strom liefert. Damit liefert eine einzelne Zelle P = U * I = 0,6V * 1A = 0,6 Watt. Ein Solarmodul hat immer eine Seriennummer.

1. Schreibe eine Klasse „solarmodule“ und eine Klasse „solarcell“. Überlege dir, welche Attribute und Methoden beide Klassen mindestens brauchen.
2. Man soll nur Solarmodule mit 120, 210 und 390 Watt erzeugen und an ihrer Seriennummer unterscheiden können. Es sind also nur bestimmte Anzahlen an Solarzellen pro Solarmodul zulässig. Seriennummer, die Anzahl der Solarzellen und die Leistung des gesamten Solarmodul sollen sich abfragen lassen.
3. einzelne Solarzellen in einem Modul können verschattet sein. Früher brach dann die Leistung des gesamten Moduls komplett ein. Ein findiger Ingenieur hat eine technische Lösung dafür entwickelt. Damit er die auch umsetzen kann, muss man herausfinden können, wie viele Solarzellen in einem Modul gerade verschattet sind. Bei Verschattung bricht die Spannung einer Solarzelle auf unter 0,3V ein. Erweitere die Klasse „solarmodule“ um eine entsprechende Methode, die die Anzahl der verschatteten Solarzellen zurückgibt.



Tipps

Du musst dir für ein Solarmodul immer sehr viele Solarzellen erzeugen. Am besten geht das mit einem Array von Solarzellen, dass du mit einer Methode erzeugen kannst. Ob du das im Hauptprogramm oder innerhalb der Klasse „solarmodule“ machst, ist im Prinzip gleichwertig.

Klicke hier für einen Lösungsansatz

solars.py

```
import random

class solarmodule:
    def __init__(self, serial, watts):
        # Wenn die Wattwerte stimmen - Objekt bauen
        if (watts == 120) or (watts == 210) or (watts == 390):
            self.watts = float(watts)
            self.cells = self.buildCellArray()
```

```
        # Sonst mit Fehlermeldung abbrechen
        else:
           print("Modulecreation not successful!")
           quit()
        self.serial = serial

    # Modul mit Zellen füllen
    def buildCellArray(self):
        # Anzahl der Zellen aus der Wattzahl errechnen
        cellCount = self.watts/0.6
        # Array für Zellen erzeugen
        cells = []
        # Array mit Zellobjekten füllen
        for i in range(int(cellCount)):
            cells.append(solarcell(i,0.6))
        return cells

    # Zufällige Zelle unter 0.6 Volt setzen
    # Count gibt die Anzahl der zu verändernden Zellen an
    def shadowCells(self, count):
        for j in range(count):
            # Wende auf eine zufällige Zelle die Methode "setCurrent"
an, die Zelle wird von der Hilfsmethode returnRandomElement()
zurückgegeben
            # Das ist eine üble Zeile, daher nochmal langsam:
            # self                              : Schlüsselwort, dass die
Methode returnRandomElement() des aktuellen Objekts aufgerufen werden
soll
            # returnRandomElement(self.cells) : Aufruf der Methode,
Übergabe des Zellarrays. Die Methode returnRandomElement() nimmt gibt
eine zufällige Zelle zurück
            # setCurrent(0.2)                   : Auf diese Zelle wird
die Methode setCurrent() der Klasse solarcell angewendet
            # Also kurz                         : "Setze die aktuelle
Spannung einer beliebigen Zelle auf 0.2V."
            self.returnRandomElement(self.cells).setCurrent(0.2)

    # Anzahl der momentan verschatteten Zellen zurückgeben
    def returnShadowedCells(self):
        count = 0
        for h in range(len(self.cells)-1):
            if self.cells[h].actualCurrent < 0.6:
                count += 1
        return count

    # Hilfsmethode: Eine zufällige Zelle zurückgeben
    def returnRandomElement(self, array):
        zufallszahl = random.randint(0,len(array)-1)
        return array[zufallszahl]
```

```
class solarcell:
    def __init__(self, id, actualCurrent):
       self.id = id
       self.actualCurrent = actualCurrent

    # Methode zum Setzen der Zellspannung (current)
    def setCurrent(self, current):
        self.actualCurrent = current

# Some Testing

# Modul erzeugen
module01 = solarmodule(1, 390)

# Modul mit zufälliger Anzahl verschatteter Zellen ausstatten
module01.shadowCells(random.randint(0,100))

# Anzahl der verschatteten Zellen ausgeben
print(f'Es gibt gerade {module01.returnShadowedCells()} verschattete
Zellen.' )
```

From:
https://schule.riecken.de/ - **Unterrichtswiki**

Permanent link:
https://schule.riecken.de/doku.php?id=informatik:algorithmisch:aufgabe:pythonobjects

Last update: **2024/07/17 07:33**