

03c_Miniuebungen

0.1 Mini-Aufgaben zur Überprüfung des Verständnis: Objektorientierung

0.1.1 Objektorientierte Modellierung mit Papier und Bleistift

Modellieren Sie objektorientiert ein Klassenzimmer (sic!). Es besteht minimal aus einer Tafel, einigen Stühlen und Tischen, sowie einem Lehrerpult. Überlegen Sie sich sinnvolle Attribute für die einzelnen Bestandteile eines Klassenzimmers, so dass bspw. Stühle und Tische bzgl. Sitzhöhe in einer Jahrgangsstufe zusammenpassen, und ein Klassenzimmer immer genau so viele Stühle besitzt, wie es Sitzplätze an den Tischen gibt.

0.1.2 Verbesserung der Vektor-Klasse

Überarbeiten Sie die Klasse Vektor, indem Sie alle Attribute "privatisieren".

Ergänzen Sie dann die Subtraktion und die komponentenweise Multiplikation. Verwenden Sie dazu passende magische Methoden.

Informieren Sie sich schließlich in der Python-Dokumentation (oder bei einem der unterwegs verlinkten Online-Tutorials) über die Funktionsweise der Methode `__getitem__(self, arg)`, um auf einen Eintrag im Vektor zugreifen zu können: `v[0]`.

Vergessen Sie nicht, Ihre Modifikationen angemessen zu testen.

```
[ ]: from math import sqrt

class Vector:
    """
    Diese Klasse repräsentiert einen beliebigen Vektor im  $R^n$ .
    """

    def __init__(self, dimension, entries=None):
        """
        Konstruktor für einen Vektor. Werden keine Einträge
        übergeben, wird der Vektor mit Nullen initialisiert.

        -----
        Argumente:
        dim : Dimension
        entries : Einträge für den Vektor
        """
```

```

assert isinstance(dimension, int)
assert dimension > 0
assert entries is None or len(entries) == dimension

self.dimension = dimension
if entries is None:
    entries = [0]*dimension

self.entries = entries
self.transposed = False

def __str__(self):
    "Gibt eine Repräsentation des Vektors auf dem Bildschirm aus."
    ret = ""
    if self.transposed:
        entries = [str(x) for x in self.entries]
        ret = "| " + " ".join(entries) + " |\n"
        return ret

    for x in self.entries:
        ret += "| " + str(x) + " |\n"

    return ret

def __add__(self, other):
    "Addiert den Vektor und den Vektor other und gibt neuen Vektor zurück"
    assert other.dimension == self.dimension, "Dimension passt nicht"
    assert other.transposed == self.transposed, "Transponierter Vektor" + \
        " und nicht transponierter Vektor übergeben."

    entries = [self.entries[i] + other.entries[i] for i in range(self.
→dimension)]
    return Vector(self.dimension, entries)

def norm(self):
    "Berechnet die euklidische Norm des Vektors"
    return sqrt(sum([x*x for x in self.entries]))

def transpose(self):
    "Transponiert den Vektor"
    self.transposed = ~self.transposed

# Anlegen eines leeren Vektors:
v1 = Vector(3)
print(v1)
v1.transpose()
print(v1)

```

```
v1.transpose()
print(v1)

# Addition zweier Vektoren:
v2 = Vector(3, [1, 2, 3])
print(v2)
v3 = Vector(3, [3, 2, 1])
v4 = v2 + v3
print(v4)
print("|v4| =", v4.norm())
```

0.2 Impressum

0.2.1 Programmierkurs Python, Dominik Göddeke <https://www.ians.uni-stuttgart.de>,
Universität Stuttgart

Version vom April 2023

Lizenziert unter der Creative Commons Namensnennung 4.0 International Lizenz



Veröffentlicht auf <https://zoerr.de>, (alle Rechte am Logo vorbehalten)



Gefördert durch die Stiftung Innovation in der Hochschullehre. (alle Rechte am Logo vorbehalten)



Gefördert mit Mitteln der Deutschen Forschungsgemeinschaft (EXC 2075 - 390740016) im Rahmen der Exzellenzstrategie.

[]: