03c_Miniuebungen_Loesungen

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: v4[0].

Vergessen Sie nicht, Ihre Modifikationen angemessen zu testen.

[9]:
```
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 dimension(self):
        "Gibt die Dimension des Vektors als int zurück"
        return self.__dimension

    def transposed(self):
        "Gibt True zurück falls der Vektor ein Zeilenvektor ist, sonst False"
        return self.__transposed

    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 __sub__(self, other):
        "Subtrahiert den übergebenen Vektor von diesem 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 __mul__(self, scalar):
        "Multipliziert den Vektor mit dem übergebenen Skalar, in-place"
        for i in range(self.__dimension):
            self.__entries[i] *= scalar
        return self

    def __getitem__(self,i):
        assert i>=0, "Negative Indizes gibt es nicht"
        assert i<= self.__dimension, "Index zu groß"
        return self.__entries[i]

    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())
print()

# Subtraktion
v5 = v4 - v3
print(v5)
```

```
v5 = v5*2
print(v5)

print(v5[2])
```

```
| 0 |
| 0 |
| 0 |

| 0 0 0 |

| 0 |
| 0 |
| 0 |

| 1 |
| 2 |
| 3 |

| 4 |
| 4 |
| 4 |

|v4| = 6.928203230275509

| 1 |
| 2 |
| 3 |

| 2 |
| 4 |
| 6 |

6
```

0.2 Impressum

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

Version vom April 2023

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.

[ ]: