cslsi-06-mueller

November 21, 2018

1 Sheet 06

- Student: Simon Müller
- Mail: s.mueller1995@gmail.com / s6siume2@uni-bonn.de

```
In [8]: import random

        class Matrix:
            def __init__(self, size=(3, 3), data=None):
                self.rows = size[0]
                self.cols = size[1]
                self.data = [[random.randrange(-59083908, 5090990) for j in range(self.cols)] fo

                if data != None:
                    assert all(len(data[x]) == len(data[x+1]) for x in range(len(data) - 1))
                    self.data = data[:]

            def __str__(self):
                maxVal = 0
                for d in self.data:
                    for v in d:
                        maxVal = abs(v) if abs(v) > maxVal else maxVal

                maxLen = len(str(maxVal)) + 2
                maxLineLen = (maxLen + 1) * self.cols
                retStr = ''

                for i in range(self.rows):
                    line = ' '.join(format(elm, str(maxLen) +'d') for elm in self.data[i])
                    line = '{:<{x}}'.format(line, x=maxLineLen)
                    retStr += line + '\n'

                return retStr

            def getRow(self, idx):
                assert idx >= 0 and idx < self.rows
                return self.data[idx]
```

```
def getCol(self, idx):
    assert idx >= 0 and idx < self.cols
    return [self.data[r][idx] for r in range(self.rows)]

def dot(self, row, col):
    assert len(row) == len(col)
    dp = 0
    for i in range(len(row)):
        dp += row[i] * col[i]
    return dp

def __add__(self, other):
    assert type(other) == type(self)
    assert other.rows == self.rows and other.cols == self.cols

    return Matrix((self.rows, self.cols) , [[self.data[r][c] + other.data[r][c] for

def __sub__(self, other):
    return self + other * -1

def __neg__(self):
    return self * -1

def __pos__(self):
    return self * 1

def __mul__(self, other):
    if type(other) == int or type(other) == float:
        return Matrix((self.rows, self.cols) , [[self.data[r][c] * other for c in ra
    elif type(other) == type(self):
        assert self.cols == other.cols and self.rows == other.rows
        return Matrix((self.rows, self.cols) , [[self.data[r][c] * other.data[r][c]
    else:
        raise Exception('Invalid type of second operand')

def __matmul__(self, other):
    assert self.cols == other.rows
    return Matrix((self.rows, other.cols), [[self.dot(self.getRow(r), other.getCol(c

def __rmul__(self, other):
    assert type(other) == int or type(other) == float
    return self * other

def __eq__(self, other):
    assert type(self) == type(other) and self.cols == other.cols and self.rows == ot
    for r in range(self.rows):
        if self.data[r] != other.data[r]:
```

```
                        return False
                return True

            def __pow__(self, other):
                assert type(other) == int and other > 0
                if other == 1:
                    return Matrix((self.rows, self.cols), self.data)
                ret = Matrix((self.rows, self.cols), self.data)
                while other > 1:
                    ret *= self
                    other -= 1
                return ret

            def __getitem__(self, key):
                r = key[0]
                c = key[1]
                assert r >= 0 and r < self.rows
                assert c >= 0 and c < self.cols

                return self.data[r][c]

            def __setitem__(self, key, value):
                r = key[0]
                c = key[1]
                assert r >= 0 and r < self.rows
                assert c >= 0 and c < self.cols

                self.data[r][c] = value
```

```
In [9]: testMat = Matrix(size=(2, 2))
        testMat2 = Matrix(size=(2, 2))
        print(testMat)
        print(testMat2)
        #print(testMat + testMat2)
        #print(testMat * -1)
        #print(testMat - testMat2)
        print(testMat * testMat2)
        print(testMat @ testMat2)
        print(testMat * 5)
        print(5 * testMat)
        print(testMat)
        #print(testMat == testMat)
        print(testMat**3 * testMat2 + testMat)
        print(testMat[1, 1])
        testMat[0, 0] = 40
        print(testMat)
```

```
   945503    -807058
-10678098    3270548
```

```
 -45185752  -15143552
 -48435777   -5368545

  -42723264073256    12221724790016
  517201973512146   -17558084112660

   -3632782759190    -9985546656046
  324086354463900   144146248211436

   4727515   -4035290
 -53390490   16352740

   4727515   -4035290
 -53390490   16352740

    945503    -807058
 -10678098    3270548

    -38193569433846281591001801       7960530188988055154598766
  58972288034802404723681620086    -187809769649838049792178092

3270548
        40    -807058
 -10678098    3270548
```

1.1 Exercise 2

```
In [10]: import math

         class Shapes:
             def __init__(self):
                 self.x = 0
                 self.y = 0
                 self.color = 'black'

             def position(self):
                 return (self.x, self.y)

             def translate(self, dx, dy):
                 self.x += dx
                 self.y += dy

             def area(self):
                 pass
```

```
    def cirumference(self):
        pass

    def set_color(self, newColor):
        self.color = newColor


class Rectangles(Shapes):
    def __init__(self, width, height):
        Shapes.__init__(self)
        self.height = height
        self.width = width

    def area(self):
        return self.height * self.width

    def cirumference(self):
        return 2 * self.height + 2 * self.width

class Ellipses(Shapes):
    def __init__(self, radiusH, radiusW):
        Shapes.__init__(self)
        self.rad1 = radiusH
        self.rad2 = radiusW

    def area(self):
        return self.rad1 * self.rad2 * math.pi

    def cirumference(self):
        return math.pi * (3 * (self.rad1 + self.rad2) / 2 - math.sqrt(self.rad1 * self.

class Triangles(Shapes):
    def __init__(self, height, side, base):
        Shapes.__init__(self)
        self.height = height
        self.base = base
        self.side = side

    def area(self):
        return self.height * self.base * 0.5

    def cirumference(self):
        return self.side * 2 + self.base

class Squares(Rectangles):
    def __init__(self, size):
        Rectangles.__init__(self, size, size)
```

```
class Circles(Ellipses):
    def __init__(self, radius):
        Ellipses.__init__(self, radius, radius)
```

```
In [11]: shapes = [Circles(10), Circles(2), Rectangles(10, 20), Squares(10), Triangles(10, 10, 1

         for shape in shapes:
             shape.translate(random.randint(-10, 10), random.randint(-10, 10))
             print("type: {}, position: {}, color: {}, area: {}, and circumference: {}".format(t
```

```
type: <class '__main__.Circles'>, position: (-3, -6), color: black, area: 314.1592653589793, and
type: <class '__main__.Circles'>, position: (7, 7), color: black, area: 12.566370614359172, and
type: <class '__main__.Rectangles'>, position: (-5, 4), color: black, area: 200, and circumferen
type: <class '__main__.Squares'>, position: (-10, 10), color: black, area: 100, and circumferenc
type: <class '__main__.Triangles'>, position: (-1, -4), color: black, area: 50.0, and circumfere
type: <class '__main__.Ellipses'>, position: (5, -5), color: black, area: 471.23889803846896, an
```

```
In [ ]:
```