Solutions to Exercises to

Programming Methods in Scientific Computing

David Bauer
Jendrik Stelzner

Letzte Änderung: November 15, 2017

Chapter 3: Python, the Fundamentals

Exercise 3.2

We extend the given class `Polynomial` by functions for the derivative and antiderivative:

```
class Polynomial:
    def __init__(self, coefficients):
        self.coeff = coefficients

    def __call__(self, x):
        s = 0
        for i in range(len(self.coeff)):
            s += self.coeff[i]*x**i
        return s

    def __add__(self, other):
        l = []
        if len(self.coeff) > len(other.coeff):
            l += self.coeff
            for i in range(len(other.coeff)):
                l[i] += other.coeff[i]
        else:
            l += other.coeff
            for i in range(len(self.coeff)):
                l[i] += self.coeff[i]
        return Polynomial(l)

    def __eq__(self, other):
        return self.coeff == other.coeff

    def derivative(self):
        coeff = []
        for i in range(1,len(self.coeff)):
            coeff.append(i * self.coeff[i])
        return Polynomial(coeff)

    def antiderivative(self):
        coeff = [0]
        for i in range(len(self.coeff)):
            coeff.append(self.coeff[i]/(i+1))
        return Polynomial(coeff)
```

For the given polynomial $p(x) = 3x^2 + 2x + 1$ we get the following results:

```
    >>> p = Polynomial([1,2,3])
    >>> p.derivative().coeff
    [2, 6]
```

```
    >>> p.antiderivative().coeff
    [0, 1.0, 1.0, 1.0]
    >>> p.antiderivative().derivative().coeff
    [1.0, 2.0, 3.0]
```

Exercise 3.4

```
class matrix():
    def __init__(self, entries):
        m = len(entries)
        if m == 0:
            raise ValueError("height must be positive")
        n = len(entries[0])
        if n == 0:
            raise ValueError("width must be positive")
        for i in range(1, m):
            if len(entries[i]) != n:
                raise ValueError("rows must have the same width")
        self.height = m
        self.width = n
        self.entries = entries

    def __getitem__(self, i):   # allows to get the rows via A[i]
        return self.entries[i]

    def __setitem__(self, i, k):    # allows to set rows via A[i]
        self.entries[i] = k

    def __str__(self):
        rows = ["["]*self.height
        for j in range(self.width): # build the output columnwise
            numbers = []            # numbers to appear in column j
            maxlen = 0              # maximal length of a number in column j
            for i in range(self.height):
                s = str(self[i][j])
                numbers.append(s)
                if len(s) > maxlen:
                    maxlen = len(s)
            for i in range(self.height):
                # pad the entries if they are too short
                rows[i] += numbers[i] + " "*(maxlen-len(numbers[i])) + " "
        s = ""
        for r in rows:
            s += r[:-1] + "]\n" # remove white space at the end of ech line
        s = s[:-1]              # remove empty line at the end
        return s

    def __mul__(self, other):
        if self.width != other.height:
            raise TypeError('matrix dimensions do not match')
        newentries = []
        for i in range(self.height):
            row = []
```

```
            for j in range(other.width):
                s = self[i][0] * other[0][j]    # s has the right type
                for k in range(1, self.width):
                    s += self[i][k] * other[k][j]
                row.append(s)
            newentries.append(row)
        return matrix(newentries)

    def __eq__(self, other):
        if self.height != other.height or self.width != other.width:
                return False
        for i in range(self.height):
            for j in range(self.width):
                if self[i][j] != other[i][j]:
                    return False
        return True
```

For the matrices

```
>>> A = matrix([[0,1],[1,0],[1,1]])
>>> B = matrix([[1,2,3,4],[5,6,7,8]])
>>> C = matrix([[1,0],[0,1],[1,0],[0,1]])
>>> A * (B * C) == (A * B) * C
True
```