

Prof. Dr. Claudia Müller-Birn, Barry Linnert

Objektorientierte Programmierung, SoSe 17

Übung 03

TutorIn: Thierry Meurers

Tutorium 10

Stefaan Hessmann, Jaap Pedersen, Mark Niehues

17. Mai 2017

1 Modulo in Python

Listing 1: Code zu Aufgabe 1

```
1 def modulo(x, y):
2     """
3     A1:
4     compute x modulo y
5
6     Parameters
7     -----
8     x : int or float
9     y : int
10
11    Returns
12    -----
13    int or float
14        x modulo y
15    """
16    while x - y >= 0:
17        x -= y
18    return x
```

2 Collatz-Folge in Python

Listing 2: Code zu Aufgabe 2

```
1 def colatz(n):
2     """
3     A2:
4     compute the colatz-sequence starting with n
5
6     Parameters
7     -----
8     n : int
9         start element for colatz-sequence
```

```

11     Returns
12     -----
13     list of int
14         colatz-sequence
15     int
16         length of the colatz-sequence
17     """
18     colatz_sequence = [n]
19     while n != 1:
20         if n % 2 == 0:
21             n = int(n / 2)
22         else:
23             n = 3 * n + 1
24         colatz_sequence.append(n)
25     return "Colatz-Folge: {} \nLänge der Colatz-Folge: {}".format(colatz_sequence, len(
        colatz_sequence))

28 if __name__ == '__main__':
29     print("Startelement der Colatz-Folge:")
30     n = int(input())
31     print(colatz(n))

```

3 Glückstage in Python

Listing 3: Code zu Aufgabe 3

```

1  """
2  AUFGABE 3:
3  """

5  def weekday_counter(start_date, end_date):
6      """
7      Count the number of times that 13.xx.xxxx was on a Monday, Tuesday, ...
8
9      Parameters
10     -----
11     start_date : str
12         first day as dd.mm.yyyy
13     end_date : str
14         last day as dd.mm.yyyy
15
16     Returns
17     -----
18     wd_counter : dict
19         number of weekdays on 13.xx.xxxx between start_date and end_date
20
21     """
22     wd_counter = {'Monday': 0, 'Tuesday': 0, 'Wednesday': 0, 'Thursday': 0,
23                  'Friday': 0, 'Saturday': 0, 'Sunday': 0} # empty dict
24     break_condition = False # break condition
25     # compute first 13.xx.xxxx since start_date
26     start_date = list(map(int, start_date.split('.')))
27     if start_date[0] <= 13:
28         start_date[0] = 13
29     else:
30         start_date[0] = 13
31         if start_date[1] != 12:
32             start_date[1] += 1
33         else:
34             start_date[1] = 1
35             start_date[2] += 1
36     # compute last 13.xx.xxxx before end_date
37     end_date = list(map(int, end_date.split('.')))
38     if end_date[0] >= 13:

```

```

39     end_date[0] = 13
40 else:
41     end_date[0] = 13
42     if end_date[1] != 1:
43         end_date[1] -= 1
44     else:
45         end_date[1] = 12
46         end_date[2] -= 1

48 date = start_date
49 while not break_condition:
50     if date == end_date:
51         break_condition = True
52         # use function from exercise 1.1
53         weekday = weekdays(date[0], date[1], date[2])
54         wd_counter[weekday] += 1
55         # go to next 13.xx.xxxx
56         if date[1] != 12:
57             date[1] += 1
58         else:
59             date[1] = 1
60             date[2] += 1

62     return wd_counter

64 def weekdays(day, month, year):
65     r"""
66     Aufgabe 1 aus Übung 1

67     Parameters
68     -----
69     day: integer value,
70         day of the month, has to be between 1 and 31
71     month: integer value,
72         month as in the gregorian calendar, has to be between 1 and 12
73     year: integer value,
74         year
75
76     Returns
77     -----
78     weekday: string
79
80     """
81
82     # check input data
83     if not (day > 0 and day <= 31):
84         raise ValueError('please, choose a day between 1 and 31.')
85     if not (month > 0 and month <= 12):
86         raise ValueError('please, choose a month between 1 and 12.')
87     if (month == 2 and day > 29):
88         raise ValueError('The Month February as maximal 29 days')
89     if (month in [4, 6, 9, 11] and day > 30):
90         raise ValueError('April, June, September and November have just 30 days')
91
92     # initialize weekday list
93     weekday = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
94
95     # Calculate Weekdays by Georg Glaeser
96     # https://de.wikipedia.org/wiki/Wochentagsberechnung
97     transformed_month = ((month - 3) % 12) + 1
98     century = int(year/100)
99     decade = year - century*100
100
101     # adapted decade and century
102
103     if (month == 1) | (month == 2):
104         decade = (decade - 1) % 100

```

```

106         if decade == 99:
107             century -= 1

109     w = (day + int(2.6 * transformed_month - 0.2) + decade + (decade//4) + (century//4) - 2
        * century) % 7

111     return weekday[w]

115 if __name__ == '__main__':
116     print("Enter start-date as dd.mm.yyyy:")
117     start_date = input()
118     print("Enter end-date as dd.mm.yyyy:")
119     end_date = input()
120     print("Weekdays for 13.xx.xxxx:")
121     print(weekday_counter(start_date, end_date))

```

4 Matrizen in Python

Listing 4: Code zu Aufgabe 4

```

1 def create_matrix(n, m):
2     """
3     Create an empty n * m matrix
4
5     Parameters
6     -----
7     n : int
8         Rows
9     m : int
10        Columns
11
12    Returns
13    -----
14    matrix : list of lists
15
16    """
17    matrix = [[None]*m for i in range(n)]
18    return matrix

21 def fill_matr(matrix):
22     """
23     Fill an empty matrix with inputs.
24
25     Parametres
26     -----
27     matrix : list of lists
28         empty matrix
29
30    Retruns
31    -----
32    matrix : list of lists
33        matrix filled with values
34
35    """
36    for j in range(len(matrix)):
37        for i in range(len(matrix[0])):
38            print("enter another element")
39            matrix[j][i] = int(input())
40    print("matrix is full")
41    return matrix

```

```

44 def make_matrix():
45     """
46     Create a matrix. Size and elements are set by user via input().
47
48     Parameters
49     -----
50     None
51
52     Returns
53     -----
54     matrix : list of lists
55         matrix with values
56
57     """
58     print("Create n*m Matrix")
59     print("Enter matrix parameters:")
60     print("n = ")
61     n = int(input())
62     print("m = ")
63     m = int(input())
64     matrix = create_matrix(n, m)
65     matrix = fill_matr(matrix)
66     return matrix
67
68
69 def transpose_matrix(matrix):
70     """
71     Transpose a give n * m matrix.
72
73     Parameters
74     -----
75     matrix : list of lists
76         matrix that will be transposed
77
78     Returns
79     -----
80     matrix_t : list of lists
81         transposed matrix
82
83     """
84     n = len(matrix[0])
85     m = len(matrix)
86     matrix_t = create_matrix(n,m)
87     for j in range(n):
88         for i in range(m):
89             matrix_t[j][i] = matrix[i][j]
90
91     return matrix_t
92
93
94 def print_matrix(matrix):
95     """
96     Prints a 2D List in a simple tab seperated way
97
98     Parameters
99     -----
100    matrix : list of lists
101        matrix to be printed
102    """
103    for i in range(len(matrix)):
104        for j in range(len(matrix[0])):
105            print(matrix[i][j], end='\t')
106        print('')
107
108
109 if __name__ == '__main__':
110     # Aufgabe a)
111     matrix = make_matrix()

```

```
112     # Aufgabe b)
113     print("Gefüllte Matrix:")
114     print_matrix(matrix)
115     # Aufgabe c)
116     print("Transponierte matrix:")
117     matrix_t = transpose_matrix(matrix)
118     print_matrix(matrix_t)
```