Freie Universität Berlin

Prof. Dr. Claudia Müller-Birn, Barry Linnert

# Objektorientierte Programmierung, SoSe 17

## Übung 03

**TutorIn: Thierry Meurers**
**Tutorium 10**

Stefaan Hessmann, Jaap Pedersen, Mark Niehues

30. Mai 2017

---

## 1 Modulo in Python

Listing 1: Code zu Aufgabe 1

```
def modulo(x, y):
    """
    A1:
    compute x modulo y

    Parameters
    ----------
    x : int or float
    y : int

    Returns
    -------
    int or float
        x modulo y
    """
    while x - y >= 0:
        x -= y
    return x
```

## 2 Collatz-Folge in Python

Listing 2: Code zu Aufgabe 2

```
def colatz(n):
    """
    A2:
    compute the colatz-sequence starting with n

    Parameters
    ----------
    n : int
        start element for colatz-sequence
```

```
    Returns
    -------
    list of int
        colatz-sequence
    int
        length of the colatz-sequence
    """
    colatz_sequence = [n]
    while n != 1:
        if n % 2 == 0:
            n = int(n / 2)
        else:
            n = 3 * n + 1
        colatz_sequence.append(n)
    return "Colatz-Folge: {} \nLänge der Colatz-Folge: {}".format(colatz_sequence, len(
    colatz_sequence))


if __name__ == '__main__':
    print("Startelement der Colatz-Folge:")
    n = int(input())
    print(colatz(n))
```

# 3 Glückstage in Python

Listing 3: Code zu Aufgabe 3

```
"""
AUFGABE 3:
"""

def weekday_counter(start_date, end_date):
    """
    Count the number of times that 13.xx.xxxx was on a Monday, Tuesday, ...

    Parameters
    ----------
    start_date : str
        first day as dd.mm.yyyy
    end_date : str
        last day as dd.mm.yyyy

    Returns
    -------
    wd_counter : dict
        number of weekdays on 13.xx.xxxx between start_date and end_date

    """
    wd_counter = {'Monday': 0, 'Tuesday' : 0, 'Wednesday' : 0, 'Thursday' : 0,
                  'Friday' : 0, 'Saturday' : 0, 'Sunday' : 0} # empty dict
    break_condition = False # break condition
    # compute first 13.xx.xxxx since start_date
    start_date = list(map(int, start_date.split('.')))
    if start_date[0] <= 13:
        start_date[0] = 13
    else:
        start_date[0] = 13
        if start_date[1] != 12:
            start_date[1] += 1
        else:
            start_date[1] = 1
            start_date[2] += 1
    # compute last 13.xx.xxxx before end_date
    end_date = list(map(int, end_date.split('.')))
    if end_date[0] >= 13:
```

```
        end_date[0] = 13
    else:
        end_date[0] = 13
        if end_date[1] != 1:
            end_date[1] -= 1
        else:
            end_date[1] = 12
            end_date[2] -= 1

    date = start_date
    while not break_condition:
        if date == end_date:
            break_condition = True
        # use function from excercise 1.1
        weekday = weekdays(date[0], date[1], date[2])
        wd_counter[weekday] += 1
        # go to next 13.xx.xxxx
        if date[1] != 12:
            date[1] += 1
        else:
            date[1] = 1
            date[2] += 1

    return wd_counter

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

    Parameters
    ----------
    day: integer value,
        day of the month, has to be be between 1 and 31
    month: integer value,
        month as in the gregorian calender, has to  be between 1 and 12
    year: integer value,
        year

    Returns
    -------
    weekday: string

    """

    # check input data
    if not (day > 0 and day <= 31):
        raise ValueError('please, choose a day between 1 and 31.')
    if not (month >0 and month <= 12):
        raise ValueError('please, choose a month between 1 and 12.')
    if (month == 2 and day > 29):
        raise ValueError('The Month February as maximal 29 days')
    if (month in [4,6,9,11] and day> 30):
        raise ValueError('April, June, September and November have just 30 days')

    # initialize weekday list
    weekday = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'
    ]

    #Calculate Weekdays by Georg Glaeser
    #https://de.wikipedia.org/wiki/Wochentagsberechnung
    transformed_month = ((month - 3) % 12) +1
    century = int(year/100)
    decade = year - century*100

    #adapted decade and century

    if (month == 1) | (month == 2):
        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

5      Parameters
6      ----------
7      n : int
8          Rows
9      m : int
10         Columns

12     Returns
13     -------
14     matrix : list of lists

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.

25     Parametres
26     ----------
27     matrix : list of lists
28         empty matrix

30     Retruns
31     -------
32     matrix : list of lists
33         matrix filled with values

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
```

```python
def make_matrix():
    """
    Create a matrix. Size and elements are set by user via input().

    Parameters
    ----------
    None

    Returns
    -------
    matrix : list of lists
        matrix with values

    """
    print("Create n*m Matrix")
    print("Enter matrix parameters:")
    print("n = ")
    n = int(input())
    print("m = ")
    m = int(input())
    matrix = create_matrix(n, m)
    matrix = fill_matr(matrix)
    return matrix


def transpose_matrix(matrix):
    """
    Transpose a give n * m matrix.

    Parameters
    ----------
    matrix : list of lists
        matrix that will be transposed

    Returns
    -------
    matrix_t : list of lists
        transposed matrix

    """
    n = len(matrix[0])
    m = len(matrix)
    matrix_t = create_matrix(n,m)
    for j in range(n):
        for i in range(m):
            matrix_t[j][i] = matrix[i][j]

    return matrix_t


def print_matrix(matrix):
    """
    Prints a 2D List in a simple tab seperated way

    Parameters
    ----------
    matrix : list of lists
        matrix to be printed
    """
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            print(matrix[i][j], end='\t')
        print('')


if __name__ == '__main__':
    # Aufgabe a)
    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)
```