III Algorithmen und Datenstrukturen

docstring-Kommentare

- #-Kommentare sind für den Leser des Programmcodes
- `docstring`-Komentare geben dem Programmieren / Anwender informationen
- Ist der erste Ausdruck in einer Funktion f oder einem Programm ein String, so wird dieser mit `help(f)` ausgegeben
- *Konvention:* Verwende den mit drei "-Zeichen eingefassten String, der über mehrere Zeilen gehen darf.
- docstring und #-Kommentare werden in Englisch geschreiben.

Beipiel: Fibonacci-Zahlen bestimmen

```
def fibo(n):
    """
    Function to calculate the first n Fibonacci-Numbers
    and print them in a list.
    """

    result = [1, 1]
    for i in range(2, n-1):
        # append the result of the values with index
        # i-2 and i-1 to the result list
        result.append(result[i-2]+result[i-1])
    return result
```

```
help(fibo)
```

```
Help on function fibo in module __main__:

fibo(n)
    Function to calculate the first n Fibonacci-Numbers and print them in a list.
```

```
fibo(15)
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
```

Doctest

- Dokumentation und Code sind gleich wichtig.
- Da auf jeden Fall der Code mit Dokumentationen versehen werden muss, ist es hilfreich auch die Test auf Korrektheit des Codes mit einzubauen.
- `doctest` = Dokumentation und Test des Programms in einem
- Syntax: “‘python “ “ ” Insert Docstring Dokumentation here.

  Now follows the dictest
  »> function_to_test(examples) expectet result “ “ ” “‘

Beispiel: Quersumme berechnen

```
def cross_sum(number):
    """
    Function to calculate the cross sum of a given number.

    >>> cross_sum(112)
    5
    >>> cross_sum(0)
    0
    >>> cross_sum(99)
    18
    """
    #convert number with type int in a string
    s_number=str(number)

    result = 0

    # create digits out of a number
    for digit in s_number:
        result = result + int(digit)
    return result
```

Doctest in Juypter:

```
import doctest

doctest.testmod(verbose=True)
```

```
Trying:
    cross_sum(112)
Expecting:
    5
**********************************************************************
File "__main__", line 5, in __main__.cross_sum
Failed example:
    cross_sum(112)
Expected:
    5
Got:
    4
Trying:
    cross_sum(0)
Expecting:
    0
ok
Trying:
    cross_sum(99)
Expecting:
    18
ok
2 items had no tests:
    __main__
    __main__.fibo
**********************************************************************
1 items had failures:
   1 of   3 in __main__.cross_sum
3 tests in 3 items.
2 passed and 1 failed.
***Test Failed*** 1 failures.

TestResults(failed=1, attempted=3)
```

Doctest im Terminal

```
!python3 -m doctest cross_number.py
```

```
**********************************************************************
```

```
File "/Users/martin/Workspace/Jupyter_Notebooks/Informatik_KS/3_Sortierverfahren/3-1_Minso
Failed example:
    cross_sum(112)
Expected:
    5
Got:
    4
**********************************************************************
1 items had failures:
   1 of   3 in cross_number.cross_sum
***Test Failed*** 1 failures.
```

Sortierverfahren

einfache Vertreter:

- Minsort
- Quicksort
- Mergesort
- Bubblesort

Problemdefinition:

Eingabe:

- Folge von n Elementen $x_1, ...x_n$
- transitiver Operator $\leq$ auf diesen Elementen
- Transitiv bedeutet: $x \leq y$ und $y \leq z \Rightarrow x \leq z$

Ausgabe:

- Folge von den n Elementen in -gemäß dem eingegebenen Operator- sortierter Reihenfolge.

Beispiel

Eingabe: 35, 14, 65, 19, 44, 8, 23, 19

Ausgabe: 8, 14, 19, 19, 23, 35, 44,65