*Dr. Christian Vollmann*
*Manuel Klar*

*Wintersemester 2022*
*[November 4, 2022]*

# Programmierung für Mathematiker

Übungsaufgaben

---

Name:

Matriculation number:

---

Allgemeine Anforderungen und Hinweise:

- bla
- bla
- bla

| Task: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Total | Grade |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Points: | | | | | | | | | | | |
| Total: | 5 | 5 | 7 | 5 | 8 | 5 | 2 | 5 | 5 | 47 | – |

**Ex** 1 5

**Half Precision: `s10e5`**
Betrachten Sie die binären Gleitkommazahlen gemäß der Parameter `s10e5`.

1. Ermitteln Sie den Verschiebewert $B$ (`BIAS`) für den Exponenten gemäß des IEEE-754 Standards, sowie den Exponentenwertebereich $e_{\text{min}}$ und $e_{\text{max}}$.
2. Mit vielen Bits wird eine Gleitkommazahl hier gespeichert? Erklären Sie die Bedeutung der einzelnen Bits und wie daraus die Gleitkommazahl gebildet wird.
3. Bestimmen Sie die relative Maschinengenauigkeit `macheps`.
4. Geben Sie die Bitmuster mit zugehörigem (ungefähren) Zahlenwert an:
   a) Signed zero und signed infinity: $\pm 0$, $\pm\infty$
   b) Kleinste und größte positive normale Zahl
   c) Kleinste und größte positive denormalisierte Zahl (*subnormals*)
   d) Kleinste und größte positive Nichtzahl (`NaN`)
   e) `0011110000000001`
   f) $\texttt{fl}(-4)$
   g) $\texttt{fl}(\frac{2}{3})$, wobei $\frac{2}{3} = (0.\overline{10})_2 = (0.10101010\ldots)_2$

**Solution:**

....

**Ex** 2 5

**Permutation**

1. Gegeben sei eine Liste von $n$ Zahlen $\texttt{perm} = [a_0, a_2, \ldots, a_{n-1}]$ mit der Eigenschaft, dass sie genau die Zahlen $0, 1, 2, \ldots, n-1$ in beliebiger Reihenfolge enthält. Die Python-Funktion `invert(perm)` gibt eine Liste zurück, in der für alle $i = 1, \ldots, n$ anstatt der Zahl $a_i$ an der Stelle $i$ die Zahl $i$ an der Stelle $a_i$ liegt. Implementieren Sie einen Test für die Funktion `invert(perm)`.
2. Implementieren Sie die Funktion `test_invert()`

**Hintergrund:** Diese Liste könnte eine Permutation darstellen. Permutationen sind fundamentale Operationen, die zum Beispiel eine wichtige Rolle in der Lösung linearer Gleichungssystem spielen.

**Solution:**

```python
#!/usr/bin/env python
# coding: utf-8
# ### Permutation
#
# (i)
def transpose(perm):
    """
    Returns the transpose of a permutaion perm (list, where i -> perm[i]).
    :param perm: list
    :return: list
    """
    permT = [[]]*len(perm)
    for i, a in enumerate(perm):
        permT[a] = i
    return permT
print("Identity")
perm = [1, 0, 2, 3]
print(perm)
print(transpose(perm))
```

```
print("Test :", [perm[k] for k in transpose(perm)], "\n")
print("Transposition")
perm = [1, 0, 2, 3]
print(perm)
print(transpose(perm))
print("Test :", [perm[k] for k in transpose(perm)], "\n")
print("Cycle")
perm = [1, 2, 3, 0]
print(perm)
print(transpose(perm))
print("Test :", [perm[k] for k in transpose(perm)], "\n")
# (ii) Die transponierte einer Permutation ist ihre Inverse. Das ermölicht hier einen
    einfachen Test.
print([perm[k] for k in transpose(perm)])
# is the short form of
newList = []
for k in transpose(perm):
    newList.append(perm[k])
print(newList)
# ### Inverse Permutation
#
# **Pseudo Code**
Input of some permutation L = (a0, ..., an-1) of length n.
Create empty list M
for all k in 0,..,n-1
    a = L[k]
    M[a] = k

print("L", L)
print("M", M)
# **Python**
L = [3,1,0,2]
def invPerm(L):
    """docstring"""
    n = len(L)  # length of list L
    M = list(range(n))  # placeholder list for result
    for k in range(n):  # iterate through the list, index by index
        a = L[k]  # a is the k-th element of L
        M[a] = k  # M_a is set to the index of a in L, i.e., k
    return M
print("L", L)
M = invPerm(L)
print("M", M)
# **Test**
#
# Da es sich um die inverse Permutation handelt sollte die Anwendung der Inversen auf die
    ursprüngliche Permutation immer die Identität ergeben.
# M is the "right inverse" of L
print([L[m] for m in M])
# M is the "left inverse" of L
print([M[l] for l in L])
```

---

**Ex** 3 7

---

**Maximum einer Liste** Die Funktion `maximum(values)` gibt das Maximale Element der Liste `values` zurück.

1. Implementieren Sie `test()` und `maximum(values)`.
2. Die Funktion `maximum` soll auch prüfen, ob die eingegebene Liste mindestens ein Element enthält und anderenfalls einen Fehler produzieren. Probieren sie aus, was der Python Befehl

   ```
   1     assert Wert "Hier gibt es ein Problem!"
   ```

   für `Wert=False` oder `True` tut und versuchen Sie ihn zu nutzen.
3. **Bonus:** Was bewirkt die (optimierte) Ausführung des Programms via `python -O` hier?

**Hintergrund:** Im Bereich des wissenschaftlichen Rechnens sind Fehlertoleranz und Performanz manchmal Gegensätze. Die elganteste Synthese dieser Gegenspieler ist dann ein Code, der sich umschalten lässt.

**Solution:**

```
#!/usr/bin/env python
# coding: utf-8
# <h1>Table of Contents<span class="tocSkip"></span></h1>
# <div class="toc"><ul class="toc-item"><li><span><a href="#Maximum" data-toc-modified-id="
    Maximum-1"><span class="toc-item-num">1  </span>Maximum</a></span></li></ul></div
    >
# ### Maximum
#
# (i)
def isMaximum(maxIndex, inputList):
    """
    Checks whether an index points to the maximum of a list.
    :param index: int
    :param inputList: list
    :return: bool
    """
    maxVal = inputList[maxIndex]
    for value in inputList:
        if maxVal < value:
            return False
    return True
# (ii)
def maximum(inputList):
    """
    Returns the value and the index of the maximal value in a list.
    :param inputList: type list
    :return: value, index
    """
    currentIndex = 0
    currentVal = inputList[currentIndex]
    for i, v in enumerate(inputList):  # enumerate iterates through and the corresponding
    indices of the items
        if v > currentVal:
            currentIndex = i
            currentVal = v
    return currentVal, currentIndex
inputList = [1, 2, 10, -3, 0]
value, index = maximum(inputList)
print("Value and index are", value, ",", index)
print("is Maximum:", isMaximum(index, inputList))
# Das funktioniert nicht nur für Zahlen
inputList = ["g", "b", "z", "d"]
maximum(inputList)
```

---

**Ex** 4 5

---

## Callback Funktion und **kwargs

1. Implementieren Sie den unten angegebenen Code. Tätigen Sie die Aufrufe `algorithm(1)` und `algorithm(1, callback=callback` Was passiert? Fügen Sie Docstrings hinzu, die kurz das Verhalten der Funktionen `algorithm` und `callback` erklären.

2. Nun soll die Funktion `callback()` nur dann etwas ausgeben, wenn `xstart` größer als ein gewisser `threshold` ist. Fügen Sie diese Eigenschaft und das Schlüsselwortargument `threshold` der Funktion `callback()` hinzu. Verändern Sie dabei die äußere Funktion `algorithm()` nicht! Welche Parameter übergeben Sie an `algorithm()`, wenn Sie nur Iterierte größer als `threshold=1.6` gedruckt bekommen möchten?

   *Hinweis:* Verwenden Sie einen optionalen Parameter: `callback(xstart, threshold=0.0)`.

3. Erläutern Sie, welchen praktischen Vorteil `**kwargs` in verschachtelten Funktionsauswertungen haben kann.

```python
from random import random

def algorithm(xstart, callback=None, **kwargs):
   for _ in range(10):
      if callback:
         callback(xstart, **kwargs)
      xstart += random() - 0.5  # der Aufruf random() generiert eine Zufallszahl in [0,1]
   return xstart

def callback(xstart):
   print(xstart)
```

**Solution:**

```python
#!/usr/bin/env python
# coding: utf-8
# ### Nested Functions with **kwargs
from random import random
def algorithm(xstart, callback = None, **kwargs):
    """
    dummy algorithm.

    :param xstart: float
    :param callback: callback function with signature callback(xstart, **kwargs)
    :param **kwargs: additional parameters for callback
    """
    for _ in range(10):
        if callback:
            callback(xstart, **kwargs)
        xstart += (random()-.5)
    return xstart
def callback(x):
    """
    callback function. Prints the iterate x whenever it is called in the outer algorithm.

    :param x: Current iterate
    """
    print(x)
xfinal = algorithm(1, callback=callback)
print("xfinal: ", xfinal)
# **Vorteil der Nutzung von \*\*kwargs**
#
# Wir definieren nun `callback()` neu, ohne `algorithm()` zu verändern. Das bedeutet,
# der Nutzer von `algorithm()` kann seine callback-Funktion nach belieben definieren und
    einsetzten und muss dazu (den eventuell schrecklich komplizierten) Algorithmus nicht
    anfassen!
def callback(x, threshold=0.0):
    """
    callback function. Prints the iterate x if its called and x > threshold.

    :param x: Current iterate
    :param threshold: threshold=0.0 (optional)
    """
    if x > threshold:
        print(x)
xfinal = algorithm(1, callback=callback, threshold=1.6)
print("xfinal: ", xfinal)
```

---

**Ex** 5 8

---

**Umwandlung: Dezimal nach Beliebig**

Schreiben Sie eine Funktion `dezToAny(d, basis)`, die eine Zahl `d` und eine Basis `basis` als Integer entgegennimmt und die Koeffizienten `a` der Darstellung der Zahl in der Basis als String zurückgibt. Es genügt hierbei, wenn Zahlensysteme bis

einschließlich der Basis 16 erlaubt sind.

**Beispiel**
a = 15, basis = 16
>>> dezToAny(a, basis) = 'f'

**Solution:**

```
#!/usr/bin/env python
# coding: utf-8
# # Umwandlung von einer Dezimalzahl in anderes Stellenwertsystem
def dezToAny(d, basis):
    """
    Converts a decimal number
    into a representation in a given basis

    :param d: Dezimal number, int, d = ex: 15
    :param basis: Basis, ex: basis = 16.

    :return:  Representation as string, ex: 'f'
    """
    digitList = '0123456789abcdefghijklmnopqrstuvwxyz'
    if basis > len(digitList):
        raise ValueError("Sorry, a basis larger than 36 is not supported.")

    c = []
    x0 = d
    xk = x0
    k = 0
    while True:
        c.append(xk % basis)
        xk = xk // basis
        k += 1
        if xk == 0:
            break
    a = ""
    for index in c:
        a += digitList[:basis][index]
    return a[::-1]
dezToAny(15,16)
```

---

**Ex** 6 5

---

**Der "moderne" Euklidische Algorithmus**

1. Schreiben Sie eine Python-Funktion, die den größten gemeinsamen Teiler zweier Zahlen mit Hilfe des "modernen" Euklidischen Algorithmus (in der iterativen Variante) bestimmt.
2. Implementieren Sie die rekursive Variante des "modernen" Euklidischen Algorithmus.
3. Testen Sie Ihren Code.

**Solution:**

```
#!/usr/bin/env python
# coding: utf-8
# ### Grösster gemeinsamer Teiler
#
# **Test**
#
# Der grösste gemeinsame Teiler ist ein Teiler beider Zahlen und es gibt keinen grösseren. Für
    Tests eignet sich `assert` es spuckt einen Fehler aus, wenn die gegebene Bedingung falsch
    ist.
def testGgt(ggt, a, b):
    """
    tests if ggt could be the greatest common divisor of a and b.
```

```
    """
    # The gcd is a divisor of both numbers.
    assert a % ggt == 0
    assert b % ggt == 0

    # There is no greater divisor of both numbers.
    # So at least one division must not return a non natural number.
    for k in range(ggt+1, min(a,b)+1):
        #print("a", a, "b", b, "k", k)
        assert not ((a % k == 0) and (b % k == 0))

    print("No error found.")
testGgt(3, 6, 9)
testGgt(3, 9, 9)
testGgt(3, 18, 9)
from math import *
def euclidAlgo(a, b, verbose=True):
    """
    computes gcd of a and b
    a, b : integer values
    verbose : optional parameter to force program to present its steps
    """
    while b != 0:
        if verbose:
            print('%s = %s * %s + %s' % (a, a//b, b, a % b))
        (a, b) = (b, a % b)
    if verbose:
        print('# Greatest common divisor is %s' % a)
    return a
numberPairs  = [[150, 304], [1000, 10], [150, 9]]
for pair in numberPairs:
    testGgt(euclidAlgo(*pair), *pair)
```

---

**Ex** 7 2

---

**Fakultät**

1. Schreiben Sie eine Python-Funktion zur Berechnung der Fakultät $n!$ einer positiven ganzen Zahl. Gehen Sie nach dem Prinzip der iterativen Programmierung vor.

2. Implementieren Sie Ihre Funktion rekursiv.

3. Schreiben Sie ein Programm, das solange eine ganze Zahl $n$ einliest bis $n$ positiv ist und dann den Wert der Fakultät $n!$ berechnet und ausgibt. Bei ungültiger Eingabe weisen Sie den Benutzer darauf hin, dass nur positive ganze Zahlen erlaubt sind. Fügen Sie die Option hinzu das Spiel bei Eingabe von `0` zu beenden.

**Solution:**

```
#!/usr/bin/env python
# coding: utf-8
def fac(n):
    """
    computes the faculty n! of a positive integer using a loop
    """
    result = 1
    for i in range(2, n+1):
        result *= i
    return result
def facRecursion(n):
    """
    computes the faculty n! of a positive integer using recursion
    """
    if n == 1:
        return 1
    else:
        return n * facRecursion(n-1)
```

```
def facCompute():
    """
    compute the faculty n! of some keyboard input value n
    """
    while True:
        n = int(input("Gib eine ganze Zahl ein: "))
        if n < 0:
            print("Negative Zahlen sind nicht erlaubt")
            continue
        elif n == 0:
            print("Das Programm wird beendet.")
            break
        else:
            print(str(n)+"! = \n",fac(n), "(iteratively),\n", facRecursion(n),"(recursively)")
if __name__ == "__main__":
    facCompute()
```

---

**Ex** 8 5

---

**SelectionSort (MaxSort)**

1. Schreiben sie eine Python-Funktion `isSorted(inputList)`, die testet, ob eine gegebene Liste von Zahlen $[a_0, \ldots, a_{n-1}]$ absteigend (also beginnend mit dem größten Wert) sortiert ist.
2. Schreibe Sie eine Python-Funktion `sort(inputList)`, die eine Liste mit Hilfe der Funktion `maximum(inputList)` sortiert. Wie oft müssen Sie die Liste durchlaufen?

**Solution:**

```
#!/usr/bin/env python
# coding: utf-8
# ### Sortiere eine Liste mit MaxSort
def maximum(inputList):
    """
    Returns the value and the index of the maximal value in a list.
    :param inputList: type list
    :return: value, index
    """
    currentIndex = 0
    currentVal = inputList[currentIndex]
    for i, v in enumerate(inputList):
        if v > currentVal:
            currentIndex = i
            currentVal = v
    return currentVal, currentIndex
# (i) Test ob eine Liste sortiert ist
def isSorted(inputList):
    """
    Checks whether a list is sorted (descending) by comparing
    all adjacency pairs
    :param inputList: list
    :return: bool
    """
    value = inputList[0]
    for nextValue in inputList[1:]:
        if value < nextValue:
            return False
        value = nextValue
    return True
# (ii) Schreibe eine Funktion, die eine Liste sortiert.
def sort(inputList):
    """
    Sorts the inputList and returns the result.
    :param inputList: list
    :return: list
    """
```

```
    newList = []
    while inputList:
        value, index = maximum(inputList)
        inputList.pop(index)  # die Methode .pop(i) entfernt das i-te Element (in-place)
        newList.append(value)  # die Methode .append() fügt ein Element zur Liste hinzu
    return newList
inputList = [0, 2, 7, 6,10]
outputList = sort(inputList)
print("Output:", outputList)
# print(inputList)
print("is Sorted:", isSorted(outputList))
```

---

**Ex** 9 5

---

**Heron's Algorithmus/ Babylonisches Wurzelziehen**
Berechnen Sie für eine beliebige nichtnegative Zahl $a \geq 0$ die Wurzel $\sqrt{a}$ bis auf einen (relativen) Fehler von $10^{-10}$ nach der folgenden iterativen Methode:

$$x_{n+1} = \frac{1}{2}\left(\frac{x_n^2 + a}{x_n}\right).$$

1. Schreiben Sie dafür eine Python Funktion `heron(a, x0, tol)`.
2. Wählen Sie verschiedene Anfangswerte $x_0$ und beobachten Sie das Konvergenzverhalten, indem Sie zum Beispiel die Fehler $|x_n - \sqrt{a}|$ in einer Liste abspeichern.
3. Erweitern Sie die Funktionsschnittstelle um einen Parameter `maxiter` (`heron(a, x0, tol, maxiter)`). Brechen Sie (zusätzlich zum Fehlertoleranzkriterium) die Iteration ab, sobald die Anzahl der Schleifendurchläufe den Wert `maxiter` erreicht hat.

*Hinweis:* Benutzen Sie die built-in Funktion `abs()` und den Python-Wert `a ** 0.5` zur Fehlermessung.

**Solution:**

```
#!/usr/bin/env python
# coding: utf-8
def heron(a, x0=0.1, tol=10e-10, maxiter =1000):
    """
    applies the iteration rule according to the so -called "Heron method"
    """
    counter = 0
    # while the error is above our tolerance we do the iteration
    while abs(x0 - a**0.5) > tol:
        # print the current error
        print("Error =", abs(x0 - a**0.5))
        # perfom one step of the heron method:
        x0 = 0.5 * ((x0 ** 2 + a) / x0)
        # update the counter
        counter += 1
        if counter > maxiter:
            # stop the while loop if too many iterations
            break
        print("\n Result: sqrt(a) =", x0, "\n")
    return x0
if __name__ == "__main__":
    a = 2.
    # Choose different initial values
    x0 = [0.1, 1., 100000 , 1.3]
    for x in x0:
        print("\n x0 =", x, "\n---------------\n")
        heron(a, x0=x)
    help(heron)
```