Institut für
Angewandte Mathematik

Leibniz
Universität
Hannover

Dr. Max Winkler
Gregor Pfau, Martin Steinbach

Hannover, 28.04.2020

Computeralgebra – Übung 2

Theorieteil

In diesem Abschnitt des Arbeitsblattes lernen Sie folgende Aspekte:

- Erstellen und Bearbeiten von Listen
- `while`-Schleifen
- Eigene Funktionen definieren

Lesen Sie sich diesen Teil bitte gründlich durch. Sie können gern nebenbei die Befehle in ihr Sage-Notebook abtippen und neue Dinge ausprobieren.

Arbeiten mit Listen

Unter http://doc.sagemath.org/html/en/thematic_tutorials/tutorial-programming-python.html#lists gibt es ein Tutorial zu Listen in SageMath. Schauen Sie sich die wichtigsten Konzepte an. Listen kann man auf verschiedene Arten anlegen:

```
In [1]: # Liste manuell anlegen
        [1,2,3,4,5]

Out[1]: [1, 2, 3, 4, 5]

In [2]: # Liste aufeinanderfolgender Zahlen von 0 bis 9(=10-1)
        [x for x in range(10)]

Out[2]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [13]: # Liste mit geraden Zahlen bis 20
         [2*x for x in range(1,11)]

Out[13]: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Mit eckigen Klammern lässt sich auf einzelne Listenelemente zugreifen.
Achtung: Die Indizierung beginnt bei 0.

```
In [4]: # Liste mit Zahlen von 1 bis 10
        L = [x for x in range(1,11)]
        print("Liste:", L)
        print("Länge der Liste:", len(L))
        print("Erstes Element:", L[0])
        print("Viertes Element:", L[4-1])
        print("Letztes Element:", L[-1])
```

```
Liste: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Länge der Liste: 10
Erstes Element: 1
Viertes Element: 4
Letztes Element: 10
```

Für Listen gibt es noch viele weitere Operationen. Speichern Sie sich die Liste als Variable ab mit

```
L = [x for x in range(10)]
```

geben Sie "L." in ein Eingabefeld ein und drücken Sie die Tabulator-Taste. Alle für Listen definierten Funktionen werden nun in einem Drop-Down-Menü angezeigt. Über

```
help(L.remove)
```

bekommen Sie den jeweiligen Hilfetext angezeigt.
Ein kurzes Beispiel:

```
In [5]: L = [x for x in range(5)]  # Liste mit Einträgen 0 bis 4 anlegen
        L.remove(3)                # 3 aus der Liste löschen
        L.append(2)                # 2 zur Liste hinzufügen
        L.sort()                   # Liste sortieren
        print("New list:", L)
        print("Element 2 occurs", L.count(2), "times")
```

```
New list: [0, 1, 2, 2, 4]
Element 2 occurs 2 times
```

Um über alle Elemente einer Liste zu iterieren bieten sich for-Schleifen an. Diese haben prinzipiell die Struktur

```
for <element> in <iterable object>:
    do something for <element>
    do something else for <element>
```

Die Einrückung bestimmt dabei den Schleifenrumpf.
In folgendem Beispiel wird die Summe aller Listeneinträge berechnet:

```
In [6]: L = [x for x in range(10)]
        s = 0
        for x in L:
            s = s+x
        print("The sum is", s)
```

```
The sum is 45
```

While-Schleifen

Merke: Die bereits bekannten for-Schleifen sind meist nur dann nützlich, wenn die Anzahl der Schleifeniterationen vorher bereits bekannt ist. Im Beispiel oben ist dies der Fall. Das würde aber nicht mehr funktionieren, wenn die Liste während der Schleife modifiziert wird.
Im folgenden Beispiel sollen alle durch 2 und 3 teilbaren Zahlen aus der Liste gelöscht werden:

```
In [7]: L = [x for x in range(13)]
        i = 0
        for x in L:
            # Teste, ob Zahl durch 2 oder 3 teilbar ist
            if x%2==0 or x%3==0:
                # Entferne Eintrag aus der Liste
                L.remove(x)
        print("The new list is", L)

The new list is [1, 3, 5, 7, 9, 11]
```

Offensichtlich ist dies falsch! Die Zahlen 3 und 9 sind durch 3 teilbar, wurden aber von der Schleife übersprungen. Das liegt daran, dass vorher die Elemente 2 bzw. 8 entfernt wurden und somit nehmen 3 und 9 ihre Plätze ein. Im nächsten Schleifendurchlauf zeigt der Iterator auf das darauffolgende Listenelement (4 und 10).
Merke: Ist die Anzahl der Schleifeniterationen vor Erreichen der Schleife nicht bekannt, dann sollte man auf while-Schleifen zurückgreifen.
Die allgemeine Syntax lautet

```
while <condition>:
    do something
    do something else
```

Die Bedingung sollte entweder True oder False ergeben. Die Einrückung bestimmt wieder den Schleifenrumpf.
Einige Beispiele für Bedingungen, die im Schleifenkopf überprüft werden können sind:

```
In [8]: print(1==3)
        print(2==2)
        print(3<5)
        print(3>=5)

False
True
True
False
```

Achtung: Falls man falsch programmiert hat, kann es auch dazu kommen, dass condition immer True ist. Dann landet man in einer Endlosschleife und Sage nimmt keine weiteren Befehle mehr an. In diesem Fall kann man den die aktuelle Berechnung über das Menü “Kernel -> Interrupt” abbrechen.
Betrachten wir das Beispiel von oben. Wir wollen die durch 2 und 3 teilbaren Elemente aus einer Liste löschen:

```
In [9]: L = [x for x in range(13)]
        i=0 # Iterationsindex (bei 0 beginnend)
        while i<len(L):
            x = L[i]

            # Der Modulooperator % gibt den Rest bei ganzzahliger Division zurück
            if x%2==0 or x%3==0:
                # Entferne Eintrag aus der Liste
                L.remove(x)
            else:
                # Springe zum nächsten Eintrag, falls nichts gelöscht wurde
                i=i+1
        print("The new list is", L)

The new list is [1, 5, 7, 11]
```

Eigene Funktionen definieren

Möchte man einzelne Code-Blöcke öfter verwenden, kann man diese auch in eine eigene Funktion auslagern. Die Syntax für Funktionen lautet

```
def <name>(<argument 1>, <argument 2>, ...):
    do something
    return <value>
```

Folgendes Beispiel berechnet die Fakultät der übergebenen Zahl:

```
In [10]: def faculty(n):
             res = 1
             for x in range(1,n+1): # Iteriert über alle Zahlen 1 bis n
                 res = res*x;       # Zwischenwert draufmultiplizieren
             return res             # Rückgabe des Ergebnisses

In [11]: print("Fakultät von 3 ist", faculty(3))
         print("Fakultät von 5 ist", faculty(5))
         print("Fakultät von 10 ist", faculty(10))

Fakultät von 3 ist 6
Fakultät von 5 ist 120
Fakultät von 10 ist 3628800
```

Hausaufgaben

Übungsaufgabe 2.1 [Berechnung von Primzahlen]

In dieser Aufgabe wollen wir über Sage Primzahlen berechnen.

Definition 1 *Eine Zahl $p \in \mathbb{N}$ heißt Primzahl, falls diese nur durch 1 und sich selbst teilbar ist.*

Eine alte Strategie um derartige Zahlen zu berechnen ist der **Sieb des Eratosthenes**. Diese Technik funktioniert wie folgt:

Algorithmus 1 (Sieb des Eratosthenes)
Input: Obere Grenze $n \in \mathbb{N}$
Output: Liste aller Primzahlen bis n.

(1) Erstelle eine Liste L der Zahlen von 2 bis n.

(2) Wähle das kleinste Element $i = 2$ in L und entferne alle Vielfachen $2i$, $3i$, ..., $k \cdot i$, ..., aus L.

(3) Wiederhole den letzten Schritt für das nächstgrößere Element i in L (also $i = 3$ im zweiten, $i = 5$ im dritten Schritt, etc.), solange $i \leq \sqrt{n}$.

Beispiel 1 *Die Umsetzung für $n = 30$ sieht wie folgt aus:*

i) Liste alle Zahlen von 2 bis 30 auf:

2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

ii) Wähle das kleinste Element der Liste (hier: 2) und streiche alle Vielfachen (4, 6, 8, ...):

2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~
11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~
21	~~22~~	23	~~24~~	25	~~26~~	27	~~28~~	29	~~30~~

iii) Wähle das nächstgrößere Element, welches noch nicht gestrichen wurde, (hier: 3) und streiche deren Vielfache (6, 9, 12, 15, ...):

2	**3**	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	25	~~26~~	~~27~~	~~28~~	29	~~30~~

iv) Wiederhole diese Schritte, bis die Liste komplett durchlaufen wurde:

2	3	~~4~~	**5**	~~6~~	**7**	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~

Die Zahlen, die übrig bleiben, sind Primzahlen. In unserem Beispiel:

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29.$$

Programmieren Sie eine Primzahlberechnung in folgender Reihenfolge:

(a) Definiere $n = 50$ und erzeuge eine Liste L mit Einträgen 2 bis n.

(b) Kopieren Sie die Liste mit L_temp = L.copy() und implementieren Sie zunächst folgenden einfachen, aber ineffizienten Algorithmus:

(i) Iteriere über alle Elemente der Liste

(ii) Nutze die vorimplementierte Funktion is_prime um festzustellen, ob die aktuelle Zahl eine Primzahl ist. Falls nicht, streiche diese aus der Liste.

Hinweis: Wir modifizieren die Liste während wir über diese Iterieren. Für welche Art von Schleife hätte man sich also im ersten Schritt entscheiden müssen?

(c) Implementieren Sie den Sieb des Eratosthenes. Schreiben Sie vielleicht zuerst einen Code, welcher den ersten Schritt (Vielfache von 2 streichen) ausführt und vervollständigen Sie den kompletten Algorithmus dann Schritt für Schritt.

Hinweise:

- Um Teilbarkeit zu überprüfen, kann man den Modulo-Operator % verwenden. Dieser gibt den Rest bei ganzzahliger Division zurück. Ist a durch b teilbar, dann liefert $a\%b = 0$. Je nach Implementierung kommen Sie aber auch ohne diesen Operator aus.
- Der L.remove-Befehl wirft eine Exception, falls das zu entfernende Element nicht in der Liste enthalten ist. Diesen Fall sollte man also vorher beispielsweise mit L.count(elem) == 0 abfangen.

(d) Testen Sie ihre Implementierungen auch für $n = 30, 100, \ldots$ und überprüfen Sie händisch, ob das Ergebnis stimmt.

Hinweis: Seien Sie ruhig kreativ und probieren auch andere Lösungswege aus. Beispielsweise kann man Schritt (c) auch so realisieren, dass man eine neue Liste aus den Vielfachen V der aktuellen Zahl erzeugt und anschließend die Differenz $L \setminus V$ über

```
[x for x in L if x not in V]
```

ermittelt.

Abgabe bis: 11.05.2020 um 12 Uhr