Praktikum Computeralgebra: Sage Grundlagen

0. Die folgenden Aufgabe stellen Sie vermutlich *mathematisch* vor keine großen Probleme. Die Blätter wurden ursprünglich für Erstsemester geschrieben und dort in einem betreuten Praktikum bearbeitet. Trotzdem ist es vielleicht hilfreich, sich auf diese Weise mit Sage vertraut zu machen (überspringen Sie einfach die Aufgaben, auf die Sie keine Lust haben). Die Schwierigkeit liegt vermutlich darin, sich an die Syntax von Sage (d.h. Python) zu gewöhnen — sollten Sie schon einmal programmiert haben, kommen Sie vermutlich selbst recht schnell dahinter; ansonsten helfe ich gerne. Vielleicht lohnt es sich auch einen Blick auf Abschnitte 3.2 und 4 des Python Tutorials unter `http://docs.python.org/tutorial/` zu werfen.

Um Sage zu starten, wechseln Sie in einer Konsole in das Verzeichnis `/home/sage/sage`, und tippen Sie `./sage` ein.

1. Die grundlegenden Rechenoperationen werden in Sage durch die Symbole `+`,`-`,`*`,`/`,`^` realisiert. Beispielsweise berechnet man $(2 \cdot 2)^2$ folgendermaßen:

```
sage: (2*2)^2
16
```

(a) Berechnen Sie den Wert der folgenden Ausdrücke mit Sage:

$$\frac{1+4}{5}, \quad \left(\frac{2}{3}\right)^{23} \cdot \left(\frac{3}{2}\right)^{23}, \quad 2^{2^{2^2}}$$

(b) Berechnen Sie $1/0$ und interpretieren Sie die letzte Zeile der Fehlermeldung.

2. Die Fakultät einer natürlichen Zahl n ist definiert als $n! = 1 \cdot 2 \cdot \cdots \cdot (n-1) \cdot n$.

(a) Berechnen Sie zunächst $m = 6!$, indem Sie die einzelnen Multiplikationen mit Sage durchführen. Vervollständigen Sie also den folgenden Ansatz:

```
sage: m= 1
sage: m= m*2
sage: m= m*3
...
sage: m
720
```

(b) Berechnen Sie nun $k = 6!$ indem Sie eine Schleife verwenden. Sollten Sie unsicher sein, wie Schleifen zu verwenden sind, dann machen Sie sich das Konzept einer Schleife zunächst noch einmal klar, indem Sie folgendes Beispiel ausführen:

```
sage: for i in range(2,10):
....:     i
....:
```

(c) Verpacken Sie ihre Implementierung in eine Funktion; vervollständigen Sie also den folgenden Ansatz:

```
sage: def fak1(n):
....:     m = 1
....:     ???
....:     return m
```

(d) Man kann die Fakultät auch durch folgende Vorschrift definieren:

$$n! := \begin{cases} 1 & \text{falls } n = 0 \\ n \cdot (n-1)! & \text{sonst} \end{cases}$$

Diese Definition erlaubt es, die Fakultätsfunktion *rekursiv* zu berechnen, indem die Berechnung für n auf die Berechnung für $n-1$ reduziert wird. Schreiben Sie nun eine Funktion, die die Fakultät rekursiv berechnet; vervollständigen Sie also den folgenden Ansatz:

```
sage: def fak2(n):
....:     if n == 0: return 1
....:     ???
```

(e) Überprüfen Sie Ihre Implementierungen, indem Sie testen, ob `fak1`, `fak2` und die eingebaute Funktion `factorial` für n zwischen 0 und 100 die gleichen Resultate liefern. (Hinweis: Es bietet sich an diesen Test mit einer Schleife zu realisieren.)

3. Das *Sieb des Eratosthenes* ist ein Algorithmus, der bei Eingabe einer natürlichen Zahl n alle Primzahlen p mit $p \leq n$ berechnet. Der Algorithmus besteht aus folgenden Schritten:

- Schreibe die Menge der Zahlen zwischen 2 und n auf und setze $k = 1$.
- Solange die Menge eine Zahl enthält die echt größer als k ist:
  - Setze k auf die nächstgrößere Zahl aus der Menge.
  - Entferne alle Vielfachen von k (also $2k, 3k, \dots$) aus der Menge.
- Gebe die resultierende Menge aus.

(a) Führen Sie den Algorithmus auf Papier für $n = 20$ durch.

(b) Führen Sie die gleichen Schritte nun mit Sage durch.
Hinweis: Das folgende Kommando erzeugt die Menge der Zahlen zwischen 2 und 20:

```
sage: M = set(range(2,21))
```

Folgendes Kommando entfernt `j*k` aus der Menge:

```
sage: M.discard(j*k)
```

(c) Machen Sie sich klar, dass der Algorithmus richtig bleibt, wenn man den Satz „*Setze k auf die nächstgrößere Zahl aus der Menge.*“ durch den Satz „*Setze k auf $k+1$.*“ ersetzt. Überzeugen Sie sich darüberhinaus, dass man genauso den Satz „*Entferne alle Vielfachen von k aus der Menge.*“ durch den Satz „*Entferne jk aus der Menge für alle j zwischen 2 und n.*“ ersetzen kann.
Führen Sie die so angepasste Version des Algorithmus für $n = 20$ durch; verwenden Sie zwei verschachtelte Schleifen für j und k, die jeweils von 2 bis 20 laufen.

(d) Der Ansatz aus (d) ist für größere n, z.B. für $n = 2000$ relativ langsam. Überlegen Sie sich, durch welche Optimierungen, die Berechnung beschleunigt werden kann. Schaffen Sie es, ihr Programm so zu verbessern, dass es die Primzahlen bis $n = 1000000$ in wenigen Sekunden berechnen kann?
Hinweis: Lange laufende Berechnungen lassen sich mit `Ctrl+c` abbrechen. Mit `len(M)` können Sie sich die Anzahl der Element einer Menge anzeigen lassen; ist Ihr Programm korrekt, sollte es 78498 Primzahlen gefunden haben.