Maple-Praktikum für Lehramt 2017 - Blatt 1

Dieses Blatt wird in Kalenderwoche 17 (ab 24. April) testiert.

Aufgaben: 4

```
> restart;
```

Herzlich willkommen im Maple-Praktikum für Lehramt!

Sie erhalten jede Woche über die Website des Lehrstuhls B für Mathematik (https://www.mathb.rwth-aachen.de) ein Worksheet wie dieses.
Ihre Aufgabe besteht jeweils darin, das Worksheet aufmerksam zu lesen, alle enthaltenen Übungsaufgaben zu bearbeiten und **den Inhalt zu verstehen**. Für inhaltliche Fragen stehen die Hiwis an allen Praktikumsterminen zur Verfügung. Sie können die Computerräume (Raum 003 im Pontdriesch 14-16 und Raum 242 im Hauptgebäude) auch zu anderen Zeiten nutzen, diese sollten dort ausgehängt sein.

Ihnen wurde oder wird über das CAMPUS-System eine Gruppe für das Praktikum zugewiesen. In dieser Gruppe findet für Sie in jeder Woche der Vorlesungszeit, außer in der ersten Woche und in der Pfingstwoche, ein Testat statt. Dabei werden Ihnen Fragen zu den Themen des Worksheets gestellt, um festzustellen, ob Sie diese verstanden haben. **Bedenken Sie, dass die Bearbeitung eines Blattes mehrere Stunden dauert.** Sie müssen vor Beginn des Testtattermins alle Aufgaben vollständig bearbeitet haben.

Bei kleineren Verständnisproblemen bekommen Sie die Möglichkeit, einen Teil des Testats in der folgenden Woche zu wiederholen. Größere Verständnisprobleme führen zum Nichtbestehen des Testats. Bis zu drei nicht bestandene Testate können an einem separaten Termin in der vorlesungsfreien Zeit wiederholt werden. Bestehen Sie vier Testate nicht oder bestehen Sie ein Testat auch bei diesem separaten Termin nicht, gilt das Praktikum als nicht bestanden.

In den Worksheets finden Sie Zeilen, die mit einem roten > als Eingabeaufforderung beginnen. In diese können Sie Maple-Code schreiben, der mit der Eingabetaste ENTER (bzw. RETURN) ausgeführt werden kann. Weiterhin finden Sie derartige Zeilen, in denen bereits Code (etwa Bespiele) steht, den Sie ausführen können **und sollen**. Ein Beispiel haben Sie bereits gesehen - mit dem r e s t a r t-Befehl wird der Speicher von Maple gelöscht, so dass zu Beginn des Worksheets keine Variablennamen bereits belegt sind. Sie können mit

```
> ?restart
```

eine Hilfeseite zum r e s t a r t-Befehl aufrufen; dies funktioniert analog für andere Befehle. Weitere Beispiele für Anweisungen, die Sie Maple geben können, sind die Grundrechenarten, in Maple +, - , * und / . Probieren Sie die folgenden Beispiele aus:

```
> 4+4;
> 3-7:
```

Anweisungen enden in Maple mit einem Doppelpunkt : oder Semikolon ; . Wie Sie an obigem Beispiel sehen, wird bei Verwendung des Doppelpunktes die Ausgabe

unterdrückt, d. h. man sieht das Ergebnis nicht, was bei längeren Rechnungen sinnvoll sein kann.

Drücken Sie die Umschalttaste SHIFT und ENTER gleichzeitig, so wird die Anweisung noch nicht sofort ausgeführt, sondern es kann eine weitere Zeile Code eingegeben werden. Selbstverständlich können Sie dies beliebig oft wiederholen und dadurch ganze Programme eingeben. Sobald Sie ENTER ohne SHIFT drücken, wird der Code ausgeführt.

```
> 3+5;  # SHIFT-ENTER
  2+7;
```

Probieren Sie es selbst aus:

```
> 7*6;
```

Steht in einer Zeile ein Rautenzeichen #, so ignoriert Maple alles, was in dieser Zeile rechts von diesem Zeichen steht. Ein Beispiel haben Sie gerade gesehen. Dies lässt sich ausnutzen, um Kommentare zu schreiben, die Ihnen, den Hiwis und Ihrem Testator das Verstehen des Codes erleichertern. **Bitte schreiben Sie Kommentare!** Beschreiben Sie mit Kommentaren den Zweck Ihres Codes, wo immer das nicht sofort ersichtlich ist.

```
> # Wenn Sie diese Zeile ausführen, passiert nichts. Probieren Sie
  es aus.
```

Wenn Sie Berechnungen durchführen, wollen Sie meistens das Ergebnis irgendwo speichern. Wie auch in vielen anderen Programmiersprachen können in Maple dazu Variablen benutzt werden. Einer Variable kann mit : = ein Wert zugewiesen werden:

```
> a:=3+4;
```

Nun kann mit a weiter gerechnet werden:

```
> b:=3*a:   # b erhaelt den Wert 3a
  b;       # Ausgabe des neuen Wertes von b
```

Beachten Sie, dass Sie durchaus mit a rechnen und das Ergebnis wieder in a speichern können:

```
> a:=3*a;
```

In einer Variable können auch andere Dinge als Zahlen gespeichert werden, zum Beispiel eine Liste oder eine Menge. Eine Liste definieren wir in der Form

```
> L:=[3,1,4,1];
```

und eine Menge in der Form

```
> M:={3,1,4,1};
```

Mit eckigen Klammern hinter dem Variablennamen können wir auf einzelne Einträge zugreifen:

```
> L[2];
  M[2];
```

Auf diese Weise können auch Einträge geändert werden:

```
> L[1]:=2:
  L;
```

Anders als bei Listen interessieren wir uns bei Mengen nicht für die Reihenfolgen und

Vielfachheiten der Elemente. Daher ist die Möglichkeit, auf einzelne Elemente einer Menge zuzugreifen, auch mit Vorsicht zu genießen, denn Maple übernimmt nicht unbedingt die Reihenfolge aus der Eingabe. Die Anzahl der Elemente lässt sich mit nops ermitteln:

```
> nops(L);
  nops(M);
```

Wir sehen, dass L vier Elemente hat, Maber nur drei.

Mit den Befehlen union, intersect und minus können die Vereinigung, der Schnitt und die Differenz von zwei Mengen A und B berechnet werden:

```
> A:={1,2,3,4,5};
  B:={3,4,5,6,7};
  A union B;
  A intersect B;
  A minus B;
```

Ist nun L eine Liste oder eine Menge, so erhält man mit op(L) die Folge aller Elemente von L.

```
> op(L);
```

Einer Liste L ein Element hinzuzufügen erreicht man z. B. durch

```
> L:=[op(L),7];
```

Um Folgen direkt ausgeben zu lassen, benutzt man den Befehl seq. Es liefert

```
> seq(2*k,k=1..4);
```

die ersten vier geraden natürlichen Zahlen. Man kann auch Summen oder Produkte von Folgen ausrechnen lassen:

```
> sum(2*k-1,k=1..5);      # Summe der ersten fuenf ungeraden
  natuerlichen Zahlen
  product(7*k, k=3..11);  # Produkt der Zahlen 7k mit k=3,...,11
```

ÜBUNG [01]:

1.) Definieren Sie die Mengen $A = \{1, 3, 4, -7, 2, 8, 12, 3, 101, 2\}$ und $B = \{2, 19, 7, 19, 101\}$ in Maple.

2.) Bilden Sie in Maple $A \cap B$, $A \cup B$ und $A \setminus B$.

3.) Definieren Sie eine Liste L in Maple, die die Quadratzahlen von 39^2 bis 77^2 enthält. Fügen Sie dann 78^2 zu L hinzu.

4.) Definieren Sie eine Liste K in Maple, die die Quadratzahlen von 1 bis 38^2 enthält. Fügen Sie dann K und L zu einer Liste zusammen.

5.) Bestimmen Sie für $M = \{a \in \mathbb{N} \mid a \text{ ungerade}, 7 \leqslant a \leqslant 27\}$ die beiden Werte $\prod_{a \in M} \frac{1}{a}$ und $\sum_{a \in M} (a^2 + 4a)$.

Man kann auch Polynomen, Funktionen, Vektoren usw. einen Namen geben. Zum Beispiel wird durch

```
> f := x^2+1;
```

das Polynom $f = x^2 + 1$ definiert. Die Funktion $g : \mathbb{R} \to \mathbb{R}$, welche von f induziert wird, erhält man mit

```
> g := x->x^2+1;
```

Jetzt kann mit

```
> g(6);
```

der Funktionswert von g an der Stelle 6 berechnet werden, während

```
> f(6);
```

kein sinnvolles Ergebnis liefert. Man beachte hier den Unterschied zwischen Polynomen und Polynomfunktionen! Dennoch kann für x ein Wert eingesetzt werden, indem man den subs-Befehl nutzt:

```
> subs(x=6,f);
```

Allerdings können trotzdem Ableitungen von f und natürlich auch von g berechnet werden, die Syntax ist aber leicht verschieden:

```
> diff(f,x);
  diff(g(x),x);
```

Maple kann auch Stammfunktionen und Integrale berechnen:

```
> int(f,x);        # unbestimmtes Integral, Stammfunktion von f
  int(g(x),x=5..9); # bestimmtes Integral, Grenzen (5 und 9)
  eingesetzt
```

In vielen Fällen kann Maple sogar Grenzwerte von Funktionen und Folgen berechnen bzw. deren Existenz überprüfen:

```
> limit(g(x), x=infinity);
  limit(g(x)/(7*x^2-4*x), x=infinity);
  limit(exp(-1/x), x=0);
```

Weiterhin können mit dem Befehl plot Graphen von Funktionen gezeichnet werden, z. B. von g im Intervall $[2, 7]$:

```
> plot(g(x),x=2..7);
```

Einige elementare Funktionen sind in Maple schon fest verankert und können ohne

eine weitere Definition aufgerufen werden, unter anderem

```
> sqrt(x); # Quadratwurzel
  exp(x);  # Exponentialfunktion
  ln(x);   # natuerlicher Logarithmus
  sin(x);  # Sinus
  cos(x);  # Cosinus
```

Zum Beispiel:

```
> exp(1);
```

Um eine numerische Näherung zu erhalten, kann evalf benutzt werden:

```
> evalf(exp(1));
```

Analog gibt es auch evalb, um bei Ausdrücken, die einen Wahrheitswert haben, diesen zu bestimmen:

```
> 5=3;
  evalb(5=3);
```

ÜBUNG [02]:

1.) Definieren Sie die Funktionen $f: \mathbb{R}^{>0} \to \mathbb{R}, x \mapsto e^{-x^2}\sqrt{\ln(x)} + \frac{3x^2+2x-7}{x^2+3}$ und $g: \mathbb{R}^{>0} \to \mathbb{R}, x \mapsto 4\sin(\ln(x)) - 2\cos(\ln(x))$ in Maple.
2.) Bestimmen Sie folgendes mit Maple:
- die Funktionswerte $f(1)$ und $g(2)$,
- die Ableitungen f' und g' sowie die Werte $f'(2)$ und $g'(5)$,
- eine Stammfunktion von g,
- das Integral $\int_1^3 f(x)\,dx$,
- den Grenzwert $\lim_{x\to\infty} f(x)$.

3.) Plotten Sie die beiden Funktionen für $x \in [0, 5]$.

Auch mit Matrizen und (Spalten-)Vektoren kann Maple umgehen. Die Eingabe erfolgt

dabei in der Form `Matrix(Zeilenzahl, Spaltenzahl, Liste der Einträge)` bzw. `Vector(Komponentenzahl, Liste der Einträge)`. Ein Spaltenvektor kann auch als Matrix mit nur einer Spalte eingegeben werden. Beispiele:

```
> M:=Matrix(2,2,[1,2,3,-1]);
  v:=Vector(2,[-1,1]);
  w:=Matrix(2,1,[-1,1]);
```

Matrizen können mit einem Punkt `.` multipliziert werden. Ebenso kann eine Matrix mit einem Vektor (der ja nichts anderes als eine Matrix mit nur einer Spalte ist) multipliziert werden. Natürlich können Matrizen nur multipliziert werden, wenn die Spalten- und Zeilenzahl zusammenpasst. Nicht alle der folgenden Beispiele führen also zu einem Ergebnis:

```
> M.M;
  M.v;
  v.M;
  M.w;
  w.M;
```

Auf die Einträge einer Matrix bzw. eines Vektors können Sie mit eckigen Klammern zugreifen:

```
> M[1,2];
  v[2];
```

Einige Befehle zur Arbeit mit Matrizen können Sie erst benutzen, wenn Sie mit

```
> with(LinearAlgebra);
```

das Paket `LinearAlgebra` geladen haben. Maple gibt Ihnen automatisch eine Liste der verfügbaren Befehle.

Der Befehl `Transpose` liefert die Transponierte einer Matrix:

```
> Transpose(M);
```

ÜBUNG [03]:

1.) Definieren Sie in Maple die Objekte

$$A=\begin{bmatrix} -1 & 2 & 7 & 11 \\ -8 & 3 & 1 & 4 \\ 0 & 1 & 2 & -3 \end{bmatrix},\ B=\begin{bmatrix} -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix},\ C=\begin{bmatrix} 1 & -2 & 0 \\ -2 & 2 & -3 \\ 0 & -3 & 3 \end{bmatrix},\ v=\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix},\ w=\begin{bmatrix} -3 \\ 1 \\ -4 \\ 1 \end{bmatrix}.$$

2.) Bestimmen Sie in Maple BA, Bv, v^{tr}, $Cv+Aw$, $v^{tr}A+4\,w^{tr}$, $A^{tr}CB^{tr}$, $v^{tr}Cv$ und $v^{tr}Aw$.

Wenn Sie möchten, dass ein Teil Ihres Codes mehrfach ausgeführt wird, können Sie eine Schleife benutzen. In Maple gibt es zwei Arten von Schleifen, die f o r-Schleife und die w h i l e-Schleife. Den zu wiederholenden Code nennen wir den *Schleifenrumpf*.

Meistens passiert nicht in jedem Durchlauf der Schleife genau das gleiche, sondern etwas leicht verschiedenes. Beispielsweise könnten Sie mit einer Variable rechnen, die in jedem Durchlauf eins größer wird, eine so genannte *Laufvariable*. Dies ist etwa von Nutzen, wenn Sie alle Einträge eines Vektors bearbeiten wollen. Für solche Fälle ist die f o r-Schleife geeignet. In der Schleife

```
> for i from 5 to 11 do
    # . . . Befehle . . .
  end do:
```

wird der Schleifenrumpf sieben Mal ausgeführt. Beim ersten Mal hat die Variable i den Wert 5, beim zweiten Mal 6 und so weiter, bis sie beim letzten Durchlauf den Wert 11 hat. Sie können auch

```
> for i from 5 by 3 to 11 do
    # . . . Befehle . . .
  end do:
```

schreiben. In diesem Fall erhöht sich der Wert von i in jedem Schritt um drei, somit hat i im ersten Durchlauf den Wert 5, im zweiten 8 und im dritten 11. Nach dem dritten Durchlauf ist die Schleife beendet.

Zum Beispiel kann im Spaltenvektor

```
> v:=Vector(4,[1,2,3,4]);
```

mit der folgenden Schleife jeder Eintrag um 3 erhöht werden:

```
> for i from 1 to 4 do
    v[i]:=v[i]+3;
  end do:
  v;
```

Sollen die Werte für i nicht regelmäßig um einen festgelegten Wert vergrößert werden, sondern aus einer bestimmten Liste oder Menge Mstammen, so kann man auch

```
> for i in M do
    # . . . Befehle . . .
  end do:
```

verwenden. Hierbei nimmt i jeden Wert aus Mgenau einmal an und die Schritte

werden mit dem jeweiligen i durchgeführt.

Bei einer while-Schleife müssen Sie eine Abbruchbedingung angegeben; in jedem Durchlauf wird zunächst geprüft, ob diese erfüllt ist.
Ist dies der Fall, wird der Schleifenrumpf ausgeführt, ansonsten ist die Schleife beendet. Definieren wir zunächst eine Variable:

```
> i:=10;
```

Ein Beispiel für eine while-Schleife mit der Abbruchbedingung $i \geq 5$ ist (bitte noch nicht ausführen!):

```
> while (i>=5) do
    # . . . Befehle . . .
  end do:
```

Hier sollte die Variable i im Schleifenrumpf verändert werden, so dass sie irgendwann einen kleineren Wert als 5 hat. Passiert dies nicht, wird die Schleife nicht beendet und läuft für immer bzw. **bis Sie auf das rote achteckige Symbol mit der Hand klicken**. Nachdem Sie dies wissen, dürfen Sie die (Endlos-)Schleife ausprobieren.

Weitere Informationen über Schleifen erhalten Sie mit

```
> ?loop
```

Mit einer if-Abfrage können Sie veranlassen, dass Code nur ausgeführt wird, wenn eine gegebene Bedingung erfüllt ist:

```
> if (i = 17) then
    # . . . Befehle . . .
  end if:
```

sorgt dafür, dass die Befehle nur ausgeführt werden, wenn $i = 17$. Mit

```
> if (i = 17) then
    # . . . Befehle . . .
  else
    # . . . Befehle . . .
  end if:
```

können Sie einen zweiten Codeblock angeben, der stattdessen ausgeführt wird, wenn die Bedingung nicht erfüllt ist.

Nun wollen wir uns noch dem Schreiben von Prozeduren widmen. Eine Prozedur ist ein Programm mit Namen, welches einmal geschrieben wird und dann auf dem gesamten Worksheet verfügbar ist. Die Syntax einer Prozedur lautet dabei wie folgt:

```
prozedurname := proc([Objekte, die der Prozedur übergeben werden])
  local [Variablen, die nur in der Prozedur verwendet werden];
  # . . . Befehle . . .
  return [Objekte, welche als Ergebnis der Prozedur zurückgegeben
werden sollen];
end proc:
```

Ein Beispiel zur besseren Illustration:

```
> facN := proc(N::integer)  # Prozedurname, zu uebergebende
  Variablen    (1)
   local i, fac;          # lokale Variablen der Prozedur
      (2)
   fac := 1;              # zunaechst fac = 1 initialisieren
      (3)
   for i from 2 to N do    #
      (4)
     fac := fac*i;         # im i-ten Schritt fac mit i
  multiplizieren  (5)
   end do;                  #
      (6)
   return fac;            # fac sollte bestimmt werden
      (7)
  end proc:                  #
      (8)
```

Das Programm berechnet zu gegebenem $N \in \mathbb{N}$ die Fakultät $N!$ und gibt das Ergebnis zurück. Betrachten wir kurz die einzelnen Schritte:
In Zeile (1) wird der Name der Prozedur (`facN`) und die Parameter festgelegt, welche zum Ausführen des Programms benötigt werden (hier nur `N` vom Typ `integer` (ganze Zahl)). Die Typfestlegung (`::integer`) ist nicht zwingend und kann (sollte aber nicht) weggelassen werden. Falls mehrere Werte an die Prozedur übergeben werden sollen, so kann man diese einfach durch Kommata voneinander trennen (z. B. `binom := proc(M::integer,N::integer)`).
In Zeile (2) werden die lokalen Variablen festgelegt. Diese Variablen werden dann nur intern von der Prozedur genutzt und wieder gelöscht, nachdem die Prozedur ihre Arbeit getan hat. Außerhalb der Prozedur sind sie nicht verfügbar. Alle Variablen, die in Ihrem Programm vorkommen und nicht an dieses übergeben werden, sollten hier aufgelistet sein.
Zeilen (3)-(6) stellen die eigentlich Rechnung dar. Hier wird `fac` zunächst auf 1 gesetzt und in der `for`-Schleife wird `fac` für `i` = 2,...,`N` mit `i` multipliziert.
Die nächste Zeile legt fest, was die Rückgabe der Prozedur sein soll (in diesem Fall der Wert von `fac` nach der Schleife) Generell kann in einer Prozedur der Befehl `return` mehr als einmal vorkommen. Sobald das Programm an einem `return` angekommen ist, gibt es das gewünschte Objekt bzw. die gewünschten Objekte zurück und ist beendet.
Zeile (8) gibt schließlich an, dass der Code der Prozedur hier zu Ende ist.

Die Prozedur kann dann z. B. mit

```
> facN(100);
```

aufgerufen werden. Sie sehen hier auch, dass Maple mit beliebig langen ganzen Zahlen rechnen kann, solange der Arbeitsspeicher reicht. Die Rechnung ist exakt und das Ergebnis keine numerische Näherung.

ÜBUNG [04]:

Schreiben Sie eine Prozedur `Schnitt`, welche den Schnitt von zwei gegebenen Mengen berechnet. Benutzen Sie **nicht** den Befehl `intersect`.

```
> SCHNITT:=proc(M::set,N::set)
    # Hier sollte Ihr Code stehen.
  end proc:
> # Mit diesen Mengen können Sie Ihren Code testen.
  A:={1,2,3,4,5};
  B:={3,4,5,6,7};
  SCHNITT(A,B);
```