Frage: Wie wird die Lösung zu „Schreiben Sie ein Prädikat dreiAlsSumme/2, das für eine gegebene Liste von Zahlen und eine konkrete Zahl überprüft, ob die konkrete Zahl als Summe dreier beliebiger unterschiedlicher Elemente der Liste berechnet werden kann.“ gefunden?

Antwort: Generell benötigen Sie eine Lösungsidee, deren Aufbau Sie der Prolog-Engine mitteilen müssen. Ein Lösungsweg kann sich dabei auch aus imperativen Ansätzen, wie in Java ergeben. Bei der konkreten Aufgabe denken Sie sicherlich sofort an drei ineinander geschachtelte Schleifen, die jeweils eine passende Position für einen Wert finden sollen. Ist die Summe der drei Werte dann der Zielwert wird das Ergebnis true zurückgegeben. Laufen alle Schleifen durch und wird so kein Ergebnis gefunden, ist das Endergebnis false.

Eine direkte Umsetzung der Schleifen führt meist nicht zum Ansatz, allerdings ist die Idee dahinter brauchbar. Es werden drei Positionen benötigt, also versuche ich erstmal ob es für das erste Element im Rest der Liste weitere Elemente gibt, die die Bedingung zusammen erfüllen. Gibt es die nicht, versuche ich das zweite Element. Die Aufgabenstellung wird so auf ein einfacheres Problem, finde zwei Elemente, die die Restsumme ergeben, heruntergebrochen. Da es sich um ein neues Problem handelt, ist ein neues Prädikat zu schreiben. Die konkrete Umsetzung sieht wie folgt aus, die zweite Zeile beschreibt, die Möglichkeit, dass es mit dem ersten Element nicht klappt.

```
dreiAlsSumme([H|T], Sum) :- zweiAlsSumme(T, Tmp), Sum is H + Tmp.
dreiAlsSumme([_|T], Sum) :- dreiAlsSumme(T, Sum).
```

Beachten Sie, dass Sie Prolog nicht mitteilen müssen, dass eine vermeintliche Lösung falsch ist.

Die gleiche Idee wird einfach für das neue Prädikat auch wieder genutzt.

```
zweiAlsSumme([H|T], Sum) :- einsAlsSumme(T, Tmp), Sum is H + Tmp.
zweiAlsSumme([_|T], Sum) :- zweiAlsSumme(T, Sum).
```

Das letzte Prädikat prüft einfach, ob ein Wert in der Liste vorkommt, dazu könnte natürlich auch ein anderes Prädikat genutzt werden. Konsequent umgesetzt führt das zu folgendem Fakt und folgender Regel.

```
einsAlsSumme([Sum|_], Sum).
einsAlsSumme([_|T], Sum) :- einsAlsSumme(T, Sum).
```

Das ist es. Wichtig, dies ist nur ein Ansatz von oft sehr vielen Möglichkeiten, die oft sogar qualitativ gleichwertig, also genau so schnell und genauso lesbar sind. Ich bin bei solchen Praktikumsaufgaben oft positiv überrascht, welche verschiedenen Lösungen von Studierenden gefunden werden.

Bei dieser Aufgabe ist ein zweiter Ansatz, es werden alle Elemente aus der Liste gestrichen, bis die drei passenden Elemente vorne stehen. Wieder muss Prolog das positive Ergebnis oder das Ende der Suche mitgeteilt werden.

```
dreiAlsSumme([A, B, C | _], Sum) :- Sum is A + B + C.
```

Listen werden typischerweise elementweise bearbeitet. Der Ansatz ist, dass es mindestens ein unerwünschtes Element in der Liste gibt. Dies kann vor A, zwischen A und B oder zwischen B und C stehen. Dieses Element wird entfernt und die Prüfung auf die kürzere Liste angewandt. Die Umsetzung sieht wie folgt aus.

```
dreiAlsSumme([_, A, B, C| Rest], Sum) :-
  dreiAlsSumme([A, B, C | Rest], Sum).
dreiAlsSumme([A, _, B, C | Rest], Sum) :-
  dreiAlsSumme([A, B, C | Rest], Sum).
dreiAlsSumme([A, B, _, C| Rest], Sum) :-
  dreiAlsSumme([A, B, C | Rest], Sum).
```

Es ist nebenbei keinesfalls dramatisch nicht sofort auf die kürzeste Lösung zu kommen. Wie bei jeder Programmentwicklung gilt, dass nach dem kurzen Kick, dass eine Lösung gefunden wurde, immer der nächste Schritt mit der Suche nach einer kürzeren, schnelleren, besser lesbaren Lösung folgt. Das folgende nicht falsche Prädikat, hab ich z. B. nach etwas Überlegung wegegeworfen.

```
% wird nicht benoetigt
dreiAlsSumme([A, B, C, _ | Rest], Sum) :-
  dreiAlsSumme([A, B, C | Rest], Sum).
```

Im nächsten Schritt stellt sich für die obige Lösung die Frage, ob wirklich immer alle drei Elemente in den Regeln stehen müssen. Das ist zu verneinen, so dass die Lösung weiter gekürzt werden kann, was sie aber für unerfahrene Personen wahrscheinlich schwerer lesbar macht. Es werden nur Möglichkeiten benötigt, das erste, das zweite und das dritte Element solange zu löschen, bis die gewünschte Lösung vorne steht.

```
dreiAlsSumme([A, B, C | _], Sum) :- Sum is A + B + C.

dreiAlsSumme([_| Rest], Sum) :-
  dreiAlsSumme(Rest, Sum).
dreiAlsSumme([A, _ | Rest], Sum) :-
  dreiAlsSumme([A | Rest], Sum).
dreiAlsSumme([A, B, _ | Rest], Sum) :-
  dreiAlsSumme([A, B | Rest], Sum).
```

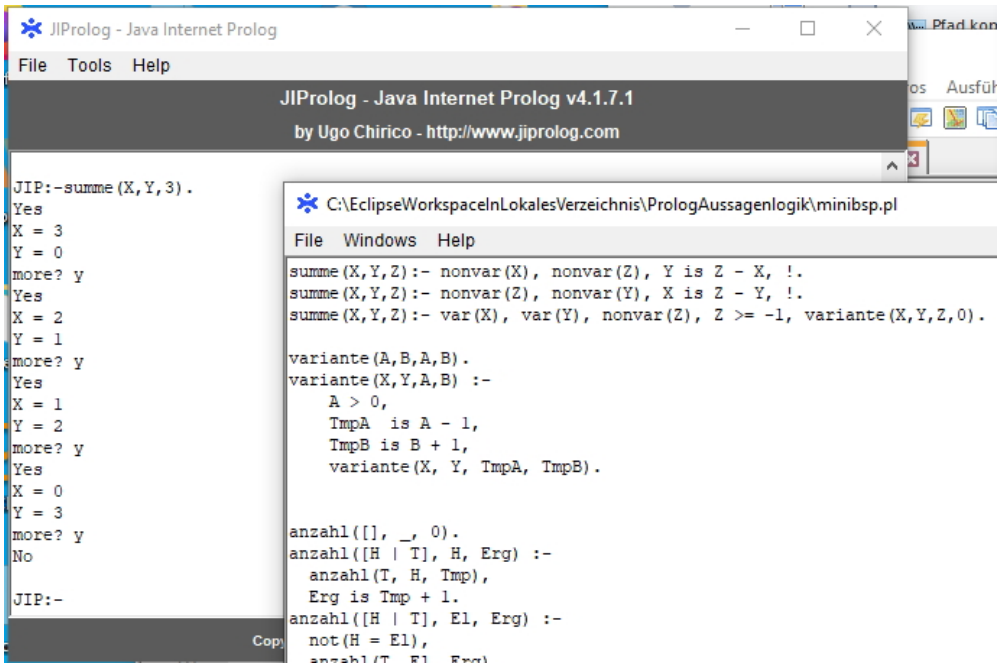
Hinweis: Die vorherigen Überlegungen deuten an, warum ich Prolog als sehr wichtiges Konzept im Informatik-Studium sehe. Durch die starke Nutzung imperativer Sprachen, wie Java, C, C++, C#, PHP und Python wird leicht die Kreativität bei der Lösungssuche eingeschränkt, was durch Prolog sich zwangsweise etwas ändern muss. Da die anderen Sprachen u. a. auch funktionale Anteile enthalten, wird der Variantenreichtum an Ideen zum Glück auch gefördert. Nebenbei kann die letzte Lösung zur vorher betrachteten Aufgabe auch in Java umgesetzt werden.

Frage: Welche Möglichkeiten gibt es eine Hausarbeit im Umfeld von Prolog zu machen?

Antwort: Lassen Sie Ihrer Kreativität freien Lauf und sprechen Sie Ihr Ergebnis mit mit ab. Sicherlich machbar sind folgende Ideen:

- ein einfaches Softwareprojekt von den Anforderungen bis zur Implementierung vollständig in Prolog
- Analyse eine der weitergehenden Funktionalität (Web, Datenbank, GUI) von SWI-Prolog (oder einem anderen Prolog) und Aufbereitung der Möglichkeiten mit eigenen Beispielen
- Entwicklung einer heterogenen Software aus Java und Prolog mit einem Prolog-Kern (sowas geht auch sehr gut mit Drools)

- Analyse der Nutzungsmöglichkeiten eines anderen Prolog-Systems, wie z. B. JIProlog (http://www.jiprolog.com/)



- oder tuProlog (https://apice.unibo.it/xwiki/bin/view/Tuprolog/) und die Analyse der Unterschiede zu SWI-Prolog