

## Verzweigungen

Neben den Schleifen spielen Verzweigungen beim Programmieren eine wesentliche Rolle. Je nach Wert oder Bedeutung von Variablen können durch Verzweigungen unterschiedliche Befehle ausgeführt werden. Als einfachste Variante stellt MuPAD ein *if*-Konstrukt zur Verfügung.

### Die IF – THEN – Anweisung

- **x:=2:** (Zeilenumbruch mit *Schift + Enter*)  
**if x = 2 then print("x ist 2") else print("x ist ungleich 2") end\_if;**

Diese zweite Eingabezeile bedeutet:

*Wenn* x = 2 ist, *dann* schreibe „x ist 2“ *ansonsten* schreibe „x ist ungleich 2“ *Ende* der *if-then*-Anweisung  
Als Ergebnis muss hier also „x ist 2“ erscheinen, da wir ja in der ersten Befehlszeile x mit 2 belegt haben.

- **x:=3: if x = 2 then print("x ist 2") else print("x ist ungleich 2") end\_if;**

Hier wird nun „x ist ungleich 2“ ausgegeben, da x mit 3 belegt wurde und 3 ja bekanntlich ungleich 2 ist.  
Nutzen wir eine *for*-Schleife, um alle geraden Zahlen von 1 bis 20 auszugeben:

- **for i from 1 to 20 do**  
    **if testtype(i, Type::Even) then print(i) end\_if**  
**end\_for;**

Diese Anweisung bedeutet:

*Für* i von 1 bis 20 tue folgendes: *Wenn* i gerade ist, *dann* schreibe i auf den Bildschirm.

## Prozeduren

Mit Hilfe von Prozeduren können mehrere Anweisungen zusammengefasst werden. Prozeduren sind im wesentlichen nichts anderes als Funktionen, die bestimmte Werte zurückliefern.

Kleiden wir unser Beispiel von oben, das testet, ob eine Zahl gleich oder ungleich 2 ist, in eine Prozedur ein:

- **GleichZwei := proc(x)                   # Kommentar: Test auf Gleichheit mit 2 #**  
**begin**  
    **if x = 2 then return("x ist 2") else return("x ist ungleich 2") end\_if;**  
**end\_proc;**

Mit *Prozedurname:=proc(Variablen)* werden der Name und die Variablen, die beim Aufruf der Prozedur benötigt werden, festgelegt. Die Anweisungen einer Prozedur werden zwischen *begin* und *end\_proc* geschrieben. Der von uns gewählte Name für die Prozedur ist *GleichZwei*. Beim Aufruf der Prozedur wird eine Variable (eine Zahl x) erwartet. Innerhalb der Prozedur steht nun die von oben bekannte *if-then*-Anweisung. Der Befehl *print* wurden durch *return* ersetzt. Bei der Abarbeitung wird die Prozedur durch *return* beendet und der bei *return* angegebene Wert wird ausgegeben. Schließt man die Prozedur mit einem Doppelpunkt hinter *end\_proc* ab, so erscheint der Prozedurtext nicht auf dem Bildschirm.

Der Aufruf der Prozedur besteht aus dem Prozedurnamen, gefolgt von den benötigten Zahlen:

- **GleichZwei(4); GleichZwei(2);**

Die folgende Prozedur liefert das Maximum zweier Zahlen zurück:

- **Maximum := proc(a, b) # Kommentar: das Maximum von a und b #**  
**begin**  
    **if a < b then return(b) else return(a) end\_if**  
**end\_proc:**

Beim Aufruf der Prozedur werden 2 Variablen (zwei Zahlen a und b) erwartet. Innerhalb der Prozedur steht eine *if-then*-Anweisung, die die beiden Zahlen a und b vergleicht und jeweils das Maximum der Zahlen zurückgibt.

- **Maximum(4, 7); Maximum(-6, 3); Maximum(3/7, 111/212);**

## Aufgaben

1. Untersuchen Sie, was die einzelnen Befehle der folgenden Prozedur bewirken und testen sie die Prozedur.

- **GeradeZahlen := proc(Anfang, Ende) local i, Liste;**  
**begin**  
    **Liste := [ ];**  
    **for i from Anfang to Ende do**  
        **if testtype(i, Type::Even) then Liste := append(Liste, i) end\_if**  
    **end\_for;**  
    **return(Liste);**  
**end\_proc:**

(*local i, Liste;* legt lokale Variablen fest. Betrachten Sie i und Liste, wie sonst, als normale Variablen.)

2. Schreiben Sie eine Prozedur, die alle negativen ganzen Zahlen einer Eingabe in eine Liste schreibt und diese ausgibt. Eingabe der Prozedur sollen wieder ein Anfangswert und ein Endwert sein (Diese Werte sollen angeben, welche ganzen Zahlen untersucht werden sollen.). Nutzen Sie *Type::NegInt*.

Speichern Sie ihre Arbeit unter *proz1.txt* ab und **drucken** Sie die Prozedur von Aufgabe 2 eventuell **aus**.